# ASYNCHRONOUS DISTRIBUTED ALTERNATING DIRECTION METHOD OF MULTIPLIERS: ALGORITHM AND CONVERGENCE ANALYSIS

*Tsung-Hui Chang*[1], *Mingyi Hong*[2], *Wei-Cheng Liao*[3], *Xiangfeng Wang*[4]

[1] School of Sci. & Eng.
The Chinese Univ. of Hong Kong, Shenzhen
Shenzhen, China 518172
E-mail: tsunghui.chang@ieee.org

[2] Dept. of IMSE & ECE
Iowa State Univ.
Ames, IA 50011, USA
E-mail: mingyi@iastate.edu

[3] Dept. of Elect. & Compt. Eng.
Univ. of Minnesota,
Twin Cities, MN 55455, USA
E-mail: liaox146@umn.edu

[4] Software Eng. Institute
East China Normal Univ.
Shanghai, China 200062
E-mail: xfwang@sei.ecnu.edu.cn

**ABSTRACT**

Alternating direction method of multipliers (ADMM) has been recognized as an efficient approach for solving many large-scale learning problems over a computer cluster. However, traditional synchronized computation does not scale well with the problem size, as the speed of the algorithm is limited by the slowest workers. In this paper, we propose an asynchronous distributed ADMM (AD-ADMM) which can effectively improve the time efficiency of distributed optimization. Our main interest lies in characterizing the convergence conditions of the AD-ADMM, under the popular *partially asynchronous model* which is defined based on a maximum tolerable delay in the network. Specifically, by considering general and possibly non-convex cost functions, we show that the AD-ADMM converges to the set of Karush-Kuhn-Tucker (KKT) points as long as the algorithm parameters are chosen appropriately according to the network delay. We also show that the asynchrony of ADMM has to be handled with care, as a slightly different implementation can significantly jeopardize the algorithm convergence.

***Index Terms***— Distributed optimization, alternating direction method of multipliers, asynchronous algorithm

## 1. INTRODUCTION

In this paper, we consider the following optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \ \sum_{i=1}^N f_i(\boldsymbol{x}) + h(\boldsymbol{x}), \tag{1}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, N$, are the cost functions to minimize and $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is a (convex) regularization function. Our interest in this paper lies in large-scale instances of problem (1), in which a large number of training samples are located across distributed computer nodes, and thereby unlikely to be solved in a centralized fashion [1]. For such large-scale and distributed scenarios, we consider a computer network with a star topology, where one master node coordinates the computation of a set of distributed workers. The star network model is often considered in the distributed optimization literature; see, e.g., [2–8]. Among these methods, the alternating direction method of multipliers (ADMM) [3, Section 7.1.1] has been recognized as an efficient method for solving large-scale problems. In particular, the distributed ADMM partitions the original learning problem into $N$ subproblems, each of which contains a partial set of training samples. At each iteration, the workers solve the subproblems locally and send the up-to-date variable information to the master; the master summarizes the distributed variable information and broadcasts the result to the workers. However, when the network size scales up, node synchronization becomes an important issue. In particular, under the synchronous protocol, the master has to wait until all the workers report their up-to-date variable information. Since the workers can have different computation and communication delays, the speed of the algorithm would be limited by the "slowest" worker.

This paper aims to generalize the existing distributed ADMM in [3] to an asynchronous network. Specifically, in the proposed asynchronous distributed ADMM (AD-ADMM), the master does not wait for all the workers to report their variable information. Instead, they make variable update whenever it receives the variable information from a partial set of workers. This would greatly reduce the waiting time of the master and speedy workers, and improve the time efficiency of the distributed optimization algorithm. In this paper, we show that, for general and possibly non-convex problems with the same form as (1), the proposed AD-ADMM is guaranteed to converge to a Karush-Kuhn-Tucker (KKT) point, as long as the algorithm parameters are chosen appropriately based on the maximum network delay. Recently, Zhang et al. [9] have considered a version of the AD-ADMM and studied its theoretical and numerical performances. However, [9] has considered the convex case only, whereas our study covers general non-convex problems. This implies that the algorithm and analysis proposed in this paper are applicable not only to standard convex learning problems but also to important non-convex problems such as the sparse PCA problem [10] and matrix factorization problems [11]. To the best of our knowledge, this is the first time that the distributed ADMM is rigorously shown convergent for non-convex problems under the asynchronous protocol.

Intriguingly, extending the distributed ADMM to an asynchronous network is by no means straightforward, and the asynchrony has to be handled with extra care. This point will be illustrated by showing that a seemingly unharmful modification of the proposed AD-ADMM can drastically jeopardize the algorithm convergence, even though such modified algorithm is equivalent to the proposed AD-ADMM under the synchronous setting. Finally, we present some numerical results to verify our theoretical claims and demonstrate the time efficiency of the proposed AD-ADMM over its synchronous counterpart.

## 2. DISTRIBUTED ADMM

In this section, we briefly review the distributed ADMM in [3, Section 7.1.1]. Consider the following consensus formulation of (1)

$$\min_{\boldsymbol{x}_0,\boldsymbol{x}_i\in\mathbb{R}^n, i\in\mathcal{V}} \ \sum_{i=1}^N f_i(\boldsymbol{x}_i) + h(\boldsymbol{x}_0) \tag{2a}$$

$$\text{s.t. } \boldsymbol{x}_i = \boldsymbol{x}_0, \ \forall i \in \mathcal{V} \triangleq \{1, \ldots, N\}. \tag{2b}$$

This work is supported in part by NSFC, China, Grant No. 61571385 and in part by NSF, Grant No. CCF-1526078.

The standard ADMM [2, 3] can be applied to solve (2), which leads to the distributed ADMM [3, Section 7.1.1] below: for iteration $k = 1, 2, \ldots,$

$$\boldsymbol{x}_0^{k+1} = \arg\min_{\boldsymbol{x}_0 \in \mathbb{R}^n} \left\{ h(\boldsymbol{x}_0) - \boldsymbol{x}_0^T \sum_{i=1}^N \boldsymbol{\lambda}_i^k + \frac{\rho}{2} \sum_{i=1}^N \|\boldsymbol{x}_i^k - \boldsymbol{x}_0\|^2 \right\}, \tag{3}$$

$$\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i \in \mathbb{R}^n} f_i(\boldsymbol{x}_i) + \boldsymbol{x}_i^T \boldsymbol{\lambda}_i^k + \frac{\rho}{2} \|\boldsymbol{x}_i - \boldsymbol{x}_0^{k+1}\|^2, \forall i \in \mathcal{V}, \tag{4}$$

$$\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \rho(\boldsymbol{x}_i^{k+1} - \boldsymbol{x}_0^{k+1}),\ \forall i \in \mathcal{V}. \tag{5}$$

Here, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $i \in \mathcal{V}$, denote the Lagrange dual variables associated with constraint (2b) and $\rho > 0$ is a penalty parameter.

As seen, the distributed ADMM in (3)-(5) is perfectly implementable in a fully parallel fashion, over a star computer network with one master node and $N$ distributed workers. Specifically, the master node is in charge of optimizing $\boldsymbol{x}_0$ by (3), and each worker $i$ is responsible for optimizing $\boldsymbol{x}_i$ and $\boldsymbol{\lambda}_i$ by (4) and (5), respectively. The master and the workers exchange the up-to-date $\boldsymbol{x}_0$ and $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i)$ with each other. Work [12] shows that the ADMM, under general convex assumptions, has a worst-case $\mathcal{O}(1/k)$ convergence rate; while [13, 14] show that the ADMM can have a linear convergence rate given strongly convex and smooth $f_i$'s. For non-convex and smooth $f_i$'s, [15] shows that (3)-(5) can converge to the set of KKT points as long as $\rho$ is large enough.

The distributed ADMM (3)-(5) is a synchronous algorithm, where at each iteration, the master has to wait until receiving the up-to-date variables $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i)$, $i \in \mathcal{V}$, from all the workers. Since the workers may have different computation and communication delays[1], the algorithm speed would be determined by the slowest worker. Therefore, under the synchronous protocol, the master and speedy workers would be idling most of the time, and thus the parallel computational resources are not fully utilized.

## 3. PROPOSED AD-ADMM

In this section, we extend the distributed ADMM to an asynchronous network. The asynchronism arises when the master does not wait for all the workers at every iteration. Instead, the master updates $\boldsymbol{x}_0$ whenever it receives $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i)$ from a subset of the workers. So neither the workers have to be synchronized with each other nor the master waits for the slowest worker. As a result, under the asynchronous protocol, both the master and speedy workers can increase the frequency of variable update. However, as a consequence of the asynchronous protocol, the master inevitably uses delayed and old $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i)$ to optimize $\boldsymbol{x}_0$, which may prevent the algorithm from convergence. To ensure the algorithm to behave properly, two additional conditions are introduced.

*Bounded delay:* To avoid the information from being too stale, it is essential to limit the maximum delay in the network. Let us denote $k$ as the iteration number of the master (i.e., the number of times for which the master updates $\boldsymbol{x}_0$), and denote $\mathcal{A}_k \subseteq \mathcal{V} \triangleq \{1, \ldots, N\}$ as the index subset of workers from which the master receives variable information during iteration $k$. We say that worker $i$ is "arrived" at iteration $k$ if $i \in \mathcal{A}_k$ and "unarrived" otherwise. We adopt the well-known "partially asynchronous" model [2], where the following bounded delay condition is assumed.

**Assumption 1** *Let $\tau \geq 1$ be a maximum tolerable delay. For all $i \in \mathcal{V}$ and iteration $k$, it must be that $i \in \mathcal{A}_k \cup \mathcal{A}_{k-1} \cdots \cup \mathcal{A}_{k-\tau+1}$.*

[1] In a heterogeneous network, the workers usually have different computation and communication capabilities. The training data can even be non-uniformly distributed across the workers. Thus, the workers can have different delays in computation and communication.

Assumption 1 implies that every worker $i$ must be arrived at least once between iteration $k - \tau + 1$ and iteration $k$. Therefore, the variable information $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i)$ used by the master will be at most $\tau$ iterations old. For this assumption to hold, at every iteration, the master would stop and wait for the workers who have been arrived for $\tau - 1$ iterations, if any. Note that, when $\tau = 1$, it corresponds to the synchronous case and the master always waits for all the workers at every iteration.

*Stepsize control:* Another intuition for an asynchronous algorithm to converge properly is to avoid the master from moving in strides. This can be achieved by adding an additional proximal term to (3), e.g., see $\frac{\gamma}{2}\|\boldsymbol{x}_0 - \boldsymbol{x}_0^k\|^2$ in (7) where $\gamma > 0$ is a penalty parameter controlling the step size of the master. Adding such proximal term is crucial in making the algorithm well-behaved in the asynchronous setting. As will be seen in the next section, a proper choice of $\gamma$ guarantees the convergence of Algorithm 1.

Based on the two conditions, the proposed AD-ADMM is presented in Algorithm 1, which includes the algorithmic steps of the master and those of the worker $i$, $i \in \mathcal{V}$. Here, $\mathcal{A}_k^c$ denotes the complementary set of $\mathcal{A}_k$, i.e., $\mathcal{A}_k \cap \mathcal{A}_k^c = \emptyset$ and $\mathcal{A}_k \cup \mathcal{A}_k^c = \mathcal{V}$. Note that in Algorithm 1 the master only update $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i)$ for all $i \in \mathcal{A}_k$. Besides, the variables $d_i$'s are introduced to count the delays of the workers. If worker $i$ is arrived at the current iteration, then $d_i$ is set to zero; otherwise, $d_i$ is increased by one. So, in Step 4 of Algorithm of the Master, the master waits until receiving messages from all the workers with $d_i \geq \tau - 1$. As a result, Assumption 1 can hold true all the time. In the next section, we present the main theoretical convergence results for Algorithm 1.

## 4. CONVERGENCE ANALYSIS

### 4.1. Convergence Conditions of Algorithm 1

We first make the following standard assumption on problem (1) (i.e., problem (2)):

**Assumption 2** *Each function $f_i$ is twice differentiable and its gradient $\nabla f_i$ is Lipschitz continuous with a constant $L > 0$; the function $h$ is convex (not necessarily smooth). Moreover, problem* (1) *is bounded below, i.e., $F^\star > -\infty$ where $F^\star$ denotes the optimal objective value of problem* (1).

It is important to note that we do not assume any convexity on $f_i$'s[2]. Our analysis will show that the AD-ADMM can converge to the set of KKT points even for non-convex $f_i$'s, as stated below.

**Theorem 1** *Suppose that Assumptions 1 and 2 hold true. Moreover, assume that*

$$\rho > \frac{(1 + L + L^2) + \sqrt{(1 + L + L^2)^2 + 8L^2}}{2}, \tag{10}$$

$$\gamma > \frac{N(1 + \rho^2)(\tau - 1)^2 - N\rho}{2}. \tag{11}$$

*Then, every limit point of $(\{\boldsymbol{x}_i^k\}_{i=1}^N, \boldsymbol{x}_0^k, \{\boldsymbol{\lambda}_i^k\}_{i=1}^N)$ generated by* (6) *and* (7) *is a KKT point of* (2).

The proof is relegated to the full paper [16]. Theorem 1 implies that the AD-ADMM converges to the set of KKT points as long as the penalty parameters $\rho$ and $\gamma$ are sufficiently large. It is worthwhile to note that a large value of $\rho$ is essential for some non-convex problems; see Section 5. Equation (11) indicates that the master should

[2] Note that for non-convex $f_i$'s, subproblem (8) is not necessarily a convex problem. However, given $\rho > L$ and Assumption 2, subproblem (8) becomes a (strongly) convex problem and is globally solvable

**Algorithm 1** Asynchronous Distributed ADMM for (2).

**Algorithm of the Master:**

**Given** initial variable $\boldsymbol{x}^0$ and broadcast it to the workers. Set $k = 0$ and $d_1 = \cdots = d_N = 0$;

**repeat**

  **wait** until receiving $\{\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{\lambda}}_i\}_{i\in\mathcal{A}_k}$ from workers $i \in \mathcal{A}_k$ and that $d_i < \tau - 1\ \forall i \in \mathcal{A}_k^c$.

  **update**

$$\boldsymbol{x}_i^{k+1} = \begin{cases} \hat{\boldsymbol{x}}_i & \forall i \in \mathcal{A}_k \\ \boldsymbol{x}_i^k & \forall i \in \mathcal{A}_k^c \end{cases}, \quad \boldsymbol{\lambda}_i^{k+1} = \begin{cases} \hat{\boldsymbol{\lambda}}_i & \forall i \in \mathcal{A}_k \\ \boldsymbol{\lambda}_i^k & \forall i \in \mathcal{A}_k^c \end{cases},$$

$$d_i = \begin{cases} 0 & \forall i \in \mathcal{A}_k \\ d_i + 1 & \forall i \in \mathcal{A}_k^c \end{cases}, \tag{6}$$

$$\boldsymbol{x}_0^{k+1} = \arg\min_{\boldsymbol{x}_0\in\mathbb{R}^n} \Big\{ h(\boldsymbol{x}_0) - \boldsymbol{x}_0^T \textstyle\sum_{i=1}^N \boldsymbol{\lambda}_i^{k+1} + \frac{\rho}{2}\sum_{i=1}^N \|\boldsymbol{x}_i^{k+1} - \boldsymbol{x}_0\|^2 + \frac{\gamma}{2}\|\boldsymbol{x}_0 - \boldsymbol{x}_0^k\|^2 \Big\}, \tag{7}$$

  **broadcast** $\boldsymbol{x}_0^{k+1}$ to the workers in $\mathcal{A}_k$.

  **set** $k \leftarrow k + 1$.

**until** a predefined stopping criterion is satisfied.

**Algorithm of the $i$th Worker:**

**Given** initial $\boldsymbol{\lambda}^0$ and set $k_i = 0$.

**repeat**

  **wait** until receiving $\hat{\boldsymbol{x}}_0$ from the master node.

  **update**

$$\boldsymbol{x}_i^{k_i+1} = \arg\min_{\boldsymbol{x}_i\in\mathbb{R}^n} f_i(\boldsymbol{x}_i) + \boldsymbol{x}_i^T\boldsymbol{\lambda}_i^{k_i} + \tfrac{\rho}{2}\|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_0\|^2, \tag{8}$$

$$\boldsymbol{\lambda}_i^{k_i+1} = \boldsymbol{\lambda}_i^{k_i} + \rho(\boldsymbol{x}_i^{k_i+1} - \hat{\boldsymbol{x}}_0). \tag{9}$$

  **send** $(\boldsymbol{x}_i^{k_i+1}, \boldsymbol{\lambda}_i^{k_i+1})$ to the master node.

  **set** $k_i \leftarrow k_i + 1$.

**until** a predefined stopping criterion is satisfied.

be more cautious in moving $\boldsymbol{x}_0$ if the network allows a larger delay $\tau$. When $\tau = 1$ (the synchronous case), $\gamma$ can be set to zero.

Let us compare Theorem 1 with the result in [9]. Firstly, the analysis in [9] is only for convex problems, whereas our convergence results also hold for both convex and non-convex problems. Secondly, the analysis in [9] is based on a specific statistical assumptions on the workers, and the final claim is that the algorithm converges *in expectation*, which is a relatively weak form of convergence. At least theoretically, it is possible that a realization of the algorithm fails to converge despite satisfying the conditions given in [9]. By contrast, our convergence results in Theorem 1 hold deterministically and thus are stronger from the theoretical point of view. One should note that (10)-(11) are sufficient conditions in general, and the algorithm may still converge in practice without exactly satisfying these conditions.

### 4.2. A Modified Scheme and Its Convergence Conditions

In Algorithm 1, the workers compute $(\boldsymbol{x}_i, \boldsymbol{\lambda}_i), i \in \mathcal{V}$, and the master is in charge of computing $\boldsymbol{x}_0$. While such distributed implementation is intuitive and natural, let us consider a modification of Algorithm 1 by moving the update of $\boldsymbol{\lambda}_i$, $i \in \mathcal{V}$ from the workers to the master. So, the workers only update $\boldsymbol{x}_i$, $i \in \mathcal{V}$, and the master, after updating $\boldsymbol{x}_0$ using $\{\boldsymbol{x}_i^{k+1}\}_{i\in\mathcal{V}}$ and $\{\boldsymbol{\lambda}_i^k\}_{i\in\mathcal{V}}$, further updates $\boldsymbol{\lambda}_i$, $i \in \mathcal{V}$, by $\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \rho(\boldsymbol{x}_i^{k+1} - \boldsymbol{x}_0^{k+1})\ \forall i \in \mathcal{V}$, and broadcasts $(\boldsymbol{x}_0^{k+1}, \boldsymbol{\lambda}_i^{k+1})$ to each worker $i \in \mathcal{A}_k$. Essentially, under the synchronous protocol, such a modification[3] does not change the convergence conditions of the distributed ADMM fundamentally. However, intriguingly, under the asynchronous protocol, such a modified AD-ADMM requires a very distinct convergence condition from Algorithm 1 and behave very differently in practice. To analyze the modified AD-ADMM, we make the following assumption.

**Assumption 3** *Each function $f_i$ is strongly convex with modulus $\sigma^2 > 0$.*

Under the strong convexity assumption, one is able to show the following convergence result.

**Theorem 2** *Suppose that Assumption 1 and Assumption 3 hold true. Moreover, let $\gamma = 0$ and*

$$0 < \rho \le \frac{\sigma^2}{(5\tau - 3)\max\{2\tau, 3(\tau - 1)\}}. \tag{12}$$

*Then, every limit point of $(\{\boldsymbol{x}_i^k\}_{i=1}^N, \boldsymbol{x}_0^k, \{\boldsymbol{\lambda}_i^k\}_{i=1}^N)$ generated by the modified AD-ADMM is a global optimal solution of* (2).

As $f_i$'s are assumed to be strongly convex, Theorem 2 somehow implies that the modified AD-ADMM may require stronger convergence conditions than Algorithm 1 in the asynchronous network. Besides, different from Theorem 1 where $\rho$ is advised to be large for Algorithm 1, Theorem 2 indicates that $\rho$ needs to be small for the modified algorithm. Interestingly and surprisingly, in some problem instances, the strongly convex $f_i$'s and a small $\rho$ seem to be *necessary* for the convergence of the modified algorithm. We illustrate this by presenting some simulation examples in the next section.

## 5. NUMERICAL RESULTS AND DISCUSSIONS

### 5.1. Example 1: Small data case (Sparse PCA and LASSO)

Theorem 1 has shown that the AD-ADMM in Algorithm 1 converges for non-convex problems. To verify this point, let us consider the following sparse PCA problem [10]

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \ -\textstyle\sum_{j=1}^N \boldsymbol{w}^T\boldsymbol{B}_j^T\boldsymbol{B}_j\boldsymbol{w} + \theta\|\boldsymbol{w}\|_1, \tag{13}$$

where $\boldsymbol{B}_j \in \mathbb{R}^{m\times n}\ \forall j = 1, \ldots, N$, and $\theta > 0$. In the simulation, each matrix $\boldsymbol{B}_j$ is a sparse random matrix with approximately 1% non-zero entries; $\theta$ is set to 0.1 and $N = 32$. We solve (13) by implementing an equivalent algorithm that is obtained by expressing Algorithm 1 from the master's point of view; see [16, Algorithm 3]. To simulate the asynchrony, at each iteration, half of the workers have a probability 0.1 to be arrived independently and half of the workers have a probability 0.8 to be arrived independently. Figure 1 shows the simulation results, where the accuracy is $|\mathcal{L}_\rho(\boldsymbol{x}^k, \boldsymbol{x}_0^k, \boldsymbol{\lambda}^k) - \hat{F}|/\hat{F}$, where $\mathcal{L}_\rho$ is an augmented Lagrangian function and $\hat{F}$ denotes the optimal objective value of (13) obtained by running the synchronous distributed ADMM for 10000 iterations (it is found in the experiments that the AD-ADMM converges to the same KKT point for different values of $\tau$). One can observe from Figure 1 that the proposed AD-ADMM indeed converges properly even though (13) is a non-convex problem. Interestingly, we note that for the example considered here, the AD-ADMM with $\gamma = 0$ works well for different values of $\tau$, even though Theorem 1 suggests that $\gamma$ should be a larger value in the worst case. However, we do observe from Figure 1 that if one sets $\beta = 1.5$ (i.e., a smaller

[3] It is equivalent to interchange the order of (3) and (4).

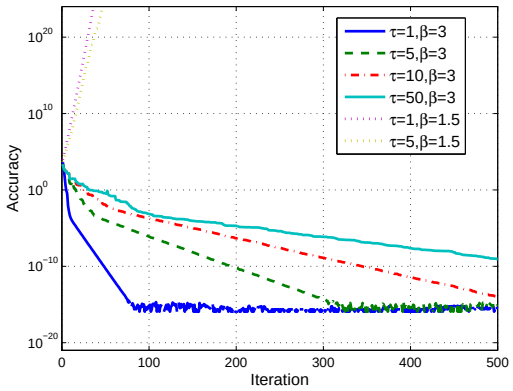



**Fig. 1**. Convergence curves of Algorithm 1 for solving the sparse PCA problem (13) with $N = 32$, $m = 1000$, $n = 500$, $\theta = 0.1$, $\rho = \beta \max_{j=1,\ldots,N} \lambda_{\max}(\boldsymbol{B}_j^T \boldsymbol{B}_j)$ and $\gamma = 0$.

value of $\rho$), then the AD-ADMM diverges even in the synchronous case ($\tau = 1$). This implies that the claim of a large enough $\rho$ is necessary for the non-convex sparse PCA problem.

Consider the following LASSO problem

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \ \sum_{i=1}^N \|\boldsymbol{A}_i\boldsymbol{w} - \boldsymbol{b}_i\|^2 + \theta\|\boldsymbol{w}\|_1, \tag{14}$$

where $\boldsymbol{A}_i \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}_i \in \mathbb{R}^m$, $i = 1, \ldots, N$, and $\theta > 0$ is a regularization parameter. The elements of $\boldsymbol{A}_i$'s are randomly generated following the Gaussian distribution with zero mean and unit variance; each $\boldsymbol{b}_i$ is generated by $\boldsymbol{b}_i = \boldsymbol{A}_i\boldsymbol{w}^0 + \boldsymbol{\nu}_i$ where $\boldsymbol{w}^0 \in \mathbb{R}^n$ is an $n \times 1$ sparse random vector with approximately $0.05n$ non-zero entries and $\boldsymbol{\nu}_i$ is a noise vector with entries following $\mathcal{N}(0, 0.01)$. To simulate an asynchronous scenario, at each iteration, half of the workers are assumed to have a probability 0.1 to be active independently, 4 workers are assumed to have a probability 0.3 to be active independently, and the remaining 4 workers are assumed to have a probability 0.8 to be active independently.

Figure 2 displays the convergence curves of AD-ADMM and the modified AD-ADMM in Section 4.2 for solving (14). Here, the accuracy is defined as $\text{accuracy} = |\mathcal{L}_\rho(\boldsymbol{x}^k, \boldsymbol{x}_0^k, \boldsymbol{\lambda}^k) - F^\star|/F^\star$ where $F^\star$ denotes the optimal objective value of problem (14). Note that, given $m = 200$ and $n = 1000$, the cost functions $f_i(\boldsymbol{w}_i) \triangleq \|\boldsymbol{A}_i\boldsymbol{w}_i - \boldsymbol{b}_i\|^2$ in (14) are not strongly convex. One can observe from Figure 2 that, under the synchronous setting (i.e., $\tau = 1$), the modified AD-ADMM exhibits a similar behavior as Algorithm 1 in Figure 2. However, the modified AD-ADMM always diverges for various values of $\rho$ even when the delay $\tau$ is as small as two. As a result, even though the two algorithms are the same under the synchronous protocol, they exhibit significantly different convergence behaviors when $\tau > 1$.

### 5.2. Example 2: Large data case (Logistic regression)

In this example, we consider the following logistic regression problem

$$\min_{\boldsymbol{w}\in\mathcal{W}} \ \sum_{j=1}^m \log\left(1 + \exp(-y_j \boldsymbol{a}_j^T \boldsymbol{w})\right) \tag{15}$$

where $y_1, \ldots, y_m$ are the binary labels of the $m$ training data, $\boldsymbol{w} \in \mathbb{R}^n$ is the regression variable and $\boldsymbol{A}_i = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m]^T \in \mathbb{R}^{m\times n}$

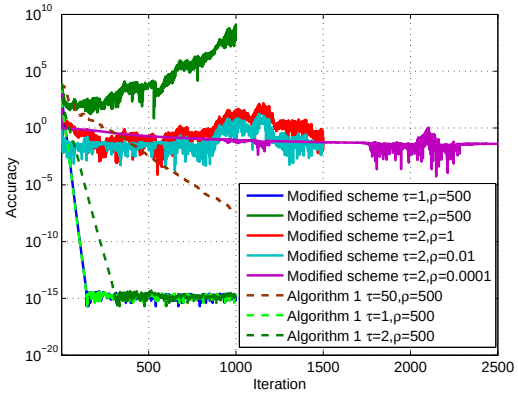
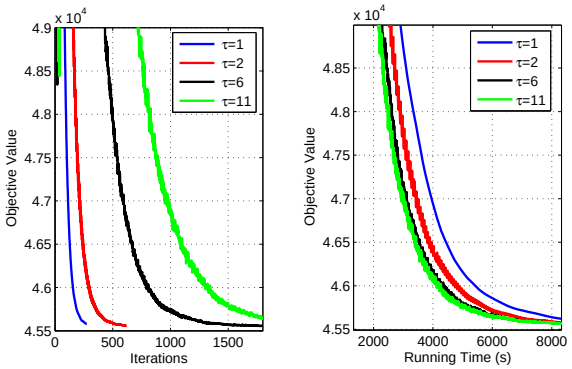



**Fig. 2**. Convergence curves of Algorithm 1 and modified AD-ADMM for solving (14) with $N = 16$, $m = 200$, $n = 1000$, $\theta = 0.1$ and $\gamma = 0$.




(a) (b)

**Fig. 3**. Convergence curves of Algorithm 1 for solving (15).

is the training data matrix. We used the MiniBooNE particle identification Data Set[4] which contains 130065 training samples ($m = 130065$) and the learning parameter has a size of 50 ($n = 50$). The constraint set $\mathcal{W}$ is set to $\mathcal{W} = \{\boldsymbol{w} \in \mathbb{R}^n \mid |w_i| \leq 10 \ \forall i = 1, \ldots, n\}$. The AD-ADMM is implemented over an HP ProLiant BL280c G6 Linux Cluster. The $n$ training samples are uniformly distributed to a set of $N = 10$ workers. For each worker, we employed the fast iterative shrinkage thresholding algorithm (FISTA) [17] to solve the corresponding subproblem (8). The stepsize of FISTA is set to 0.0001 and the stopping condition is that the 2-norm of the gradient is less than 0.001. The penalty parameters of the AD-ADMM method are set to $\rho = 0.01$ and $\gamma = 0$. Figure 3(a) and Figure 3(b) respectively show the convergence curves (objective value) of the AD-ADMM versus the iteration number and the running time (second), for various values $\tau$. One can observe from Figure 3(a) that, in terms of the iteration number, the convergence speed of AD-ADMM slows down when $\tau$ increases. However, as seen from Figure 3(b), the running time of the AD-ADMM actually is smaller than its synchronous counterpart ($\tau = 1$). Specifically, for achieving the objective value $4.6 \times 10^4$, the running time for $\tau = 11$ is about 25% faster than the synchronous algorithm ($\tau = 1$).

[4] `https://archive.ics.uci.edu/ml/datasets/MiniBooNE+particle+identification`

## 6. REFERENCES


[1] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling up Machine Learning- Parallel and Distributed Approaches*. Cambridge University Press, 2012.

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: Numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[4] F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Proc. Advances in Neural Information Processing Systems (NIPS)*, vol. 24, pp. 693-701, 2011, [Online] http://arxiv.org/abs/1106.5730.

[5] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," *Proc. Advances in Neural Information Processing Systems (NIPS)*, vol. 24, pp. 873-881, 2011, [Online] http://arxiv.org/abs/1104.5525.

[6] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," [Online] http://www.cs.cmu.edu/~muli/file/ps.pdf.

[7] M. Li, D. G. Andersen, and A. Smola, "Distributed delayed proximal gradient methods," [Online] http://www.cs.cmu.edu/~muli/file/ddp.pdf.

[8] J. Liu and S. J. Wright, "Asynchronous stochastic coordinate descent: Parallelism and convergence properties," *SIAM J. Optim.,*, vol. 25, no. 1, pp. 351–376, Feb. 2015.

[9] R. Zhang and J. T. Kwok, "Asynchronous distributed ADMM for consensus optimization," in *Proc. 31th ICML, , 2014.*, Beijing, China, June 21-26, 2014, pp. 1–9.

[10] P. Richtárik, M. Takáč, and S. D. Ahipasaoglu, "Alternating maximization: Unifying framework for 8 sparse PCA formulations and efficient parallel codes," [Online] http://arxiv.org/abs/1212.4137.

[11] Q. Ling, Y. Xu, W. Yin, and Z. Wen, "Decentralized low-rank matrix completion," in *Proc. IEEE ICASSP*, Kyoto, Japan, March 25-30, 2012, pp. 2925–2928.

[12] B. He and X. Yuan, "On the $o(1/n)$ convergence rate of Douglas-Rachford alternating direction method," *SIAM J. Num. Anal.*, vol. 50, 2012.

[13] W. Deng and W. Yin, "On the global and linear convergence of the generalized alternating direction method of multipliers," Rice CAAM technical report 12-14, 2012.

[14] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in decentralized consensus optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 7, pp. 1750–1761, April 2014.

[15] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," technical report; available on http://arxiv.org/pdf/1410.1390.pdf.

[16] T.-H. Chang, M. Hong, W.-C. Liao, and X. Wang, "Asynchronous distributed ADMM for large-scale optimization- Part I: Algorithm and convergence analysis," submitted for publication.

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