2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

A Fully Asynchronous Newton Tracking Method for Decentralized Optimization

Zhaoye Pan and Huikang Liu

***Abstract*— We consider fully asynchronous decentralized optimization over a directed graph. While various algorithms have been proposed, real-world applications require relaxing the assumption and considering communication networks with asynchronous and heterogeneous nodes. To meet these challenges, we propose an efficient and robust Newton tracking mechanism for fully asynchronous optimization. Our proposed mechanism can be adapted to different asynchronous first-order methods as required by the practical context. Through the theoretical analysis we demonstrate the R-linear rate of our method and derive an explicit expression of decaying factor under local conditions. Furthermore, numerical comparison with existing algorithms support the efficiency and robustness of our method.**

I. INTRODUCTION

In this paper, we consider a fully asynchronous decentralized optimization problem, where n agents cooperate to minimize the average of local functions

$$x^* = \arg\min_{x\in\mathbb{R}^d} \; F(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{1}$$

where $x \in \mathbb{R}^d$ is the decision variable and $f_i : \mathbb{R}^d \to \mathbb{R}$ is the loss function privately owned by node i. The optimization problem (1) has applications in various fields, such as deep learning [1], sensor networking [2], statistical learning [3], and optimal transport [4]. To leverage the benefits of decentralized storage of data and parallel computing power of nodes, decentralized algorithms for solving problem (1) have been extensively studied, especially for the design of synchronous algorithms, such as DGD [5], EXTRA [6], gradient tracking [7] based on undirected graphs, and Push-Sum [8] and Push-Pull [9] based on directed graphs.

However, due to the heterogeneity of data and systems, and the impact of uncontrollable factors such as communication delays or errors, it is essential to design asynchronous algorithms. Asynchronous algorithms can reduce waiting time, alleviate communication costs, and make the algorithm more fault-tolerant. This paper considers the *fully* asynchronous scenario: agents can use local and neighbor information (possibly stale) for local computation at any time and send the results to neighbors without any synchronous coordination and scheduling. The difference between synchronous and our fully asynchronous settings is depicted in Fig. 1. Specifically, in the synchronous setting, all nodes perform computation and communication at the same time per iteration, while in the fully asynchronous setting, nodes can start at any time. We can observe that asynchronous algorithms are more efficient, as they can perform more iterations within the same time period.

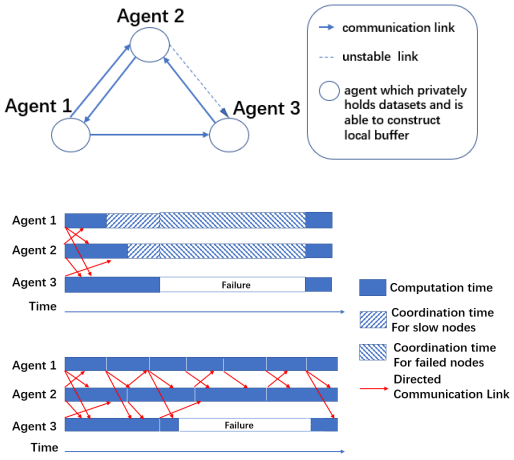


Fig. 1. Comparison of Synchronous and Asynchronous Setting.

Asynchronous Decentralized Methods. In recent years, there has been a noticeable shift in research focus towards asynchronous decentralized optimization methods. Some of these methods utilize random activation mechanisms that satisfy specific distributions, where a randomly selected group of agents are activated per iteration to perform computation and communicate with their neighbors [10]–[12]. Some works propose asynchronous algorithms that are robust to communication delays and investigate the effect of delay on convergence [13], [14]. However, these algorithms still require some degree of coordination among nodes and are not fully asynchronous. Several algorithms have been developed based on a fully asynchronous model [15]–[18], where agents can use local and neighbor information (possibly delayed) for local computation at any time and send the results to neighbors without any form of coordination or centralized scheduling. The asynchronous Gradient-Push (AGP) method proposed in [15] uses the buffer mechanism to store delayed information and is proven to converge to a neighborhood of the optimal solution. However, as shown in [18], AGP cannot converge precisely due to the uneven update frequency of each node. To tackle this issue, [18] introduces an auxiliary variable to adjust the update frequency of each node and proposes a subgradient-type method with exact convergence. However, the decaying stepsize rule makes convergence rate sub-linear. To improve the convergence rate, [16] proposes an

Z. Pan, H. Liu are with Department of Information Management and Engineering Management, Shanghai University of Finance and Economics, Shanghai, China zhaoyepan@163.sufe.edu.cn, liuhuikang@shufe.edu.cn

asynchronous algorithm over a directed network, where each node estimates the global gradient using the idea of gradient tracking [19]. Inspired by Push-Sum [20], the method in [16] gets a linear convergence approaching to the optimal point. The APPG method proposed in [17] is based on the Push-Pull algorithm over a directed network. It uses an augmented graph to model the communication of the asynchronous algorithm and shows that APPG converges to the optimal solution at a linear rate. Apart from first-order methods, some works study second-order methods that use curvature information to accelerate convergence. The work [21] adapts an approximate second-order information scheme under a partial asynchronous setting and can handle uncoordinated activation and packet loss. However, the work [21] requires that only one agent wakes up at each iteration, which requires a global clock. [12], [22] studied the asynchronous decentralized second-order algorithm, while they assume the activation of the node follows Poisson Process, this assumption make convergence analysis easier, but there is a strong dependence of the delays on the activation index which often contrasts the case in reality. To the best of our knowledge, there is currently no known decentralized second-order algorithm specifically designed for a fully asynchronous setting.

A. Major Contribution

In this paper, we propose an efficient and robust asynchronous newton tracking mechanism for fully asynchronous decentralized optimization. The main contributions are summarized as follows:

- We propose a novel newton-tracking mechanism for fully asynchronous optimization, which can be easily adapted in existing asynchronous first-order algorithms. Our mechanism enhances the performance and robustness of existing asynchronous first-order algorithms by incorporating global second-order information.
- Through our theoretical analysis, we have clearly shown that our algorithm enjoys a linear convergence rate and derived the explicit expression for decaying factor that surpasses that of existing asynchronous first-order algorithms under local conditions.
- Our numerical comparisons have further demonstrated the considerable improvements in efficiency and robustness of our mechanism when compared to both asynchronous first-order algorithms and synchronous second-order algorithms.

B. Notation

We use k to denote the index of iteration and $t(k)$ to denote the time of k-th iteration. We use $\|\cdot\|$ to denote the Euclidean norm of a vector or the largest singular value of a matrix, $\|\cdot\|_F$ to denote the Frobenius norm. We use $[n]$ to denote the numbers from 1 to n. For the aggregated variable, we define $\boldsymbol{x} = [x_1; \ldots; x_n] \in \mathbb{R}^{nd}$, $\nabla f(\boldsymbol{x}^k) = [\nabla f_1(x_1^k); \ldots; \nabla f_n(x_n^k)]$, $\boldsymbol{H}^k = [H_1^k; \ldots; H_n^k]$ other aggregated variables are defined similarly. For the average variable, the average decision variable over all nodes at the iteration k is defined as $\bar{\boldsymbol{x}}^k = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i^k$, $\overline{\nabla f}(\boldsymbol{x}^k) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(x_i^k) \in \mathbb{R}^d$, $\nabla F(\bar{x}^k) = \frac{1}{n}\nabla f_i(\bar{\boldsymbol{x}}^k) \in \mathbb{R}^d$, $\bar{H}^k = \frac{1}{n}\sum_{i=1}^n H_i^k$, other average variables are defined similarly. Given a buffer $\mathcal{B}$, $\mathbf{avg}\,(\mathcal{B})$ and $\mathbf{sum}\,(\mathcal{B})$ return the average and return the sum of variables in the buffer, respectively.

C. Organization

The remaining sections of this document are structured in the following manner. We provide the problem setting and proposed asynchronous newton tracking mechanism along with its implementations in Section II. We establish the convergence analysis in Section III and present the numeric examples in Section IV. Section V serves as the concluding part of the paper.

II. Problem Setting And Algorithm Development

In this section, we give a detailed description of the problem setting, then we develop a newton tracking mechanism for asynchronous decentralized optimization and adapt the mechanism over existing asynchronous first-order methods.

A. Problem Setting

We consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, n\}$ is the set of n nodes with $|\mathcal{V}| = n$ and $\mathcal{E}$ is the set of all edges. If $(i, j) \in \mathcal{E}$, it indicates that i can send message to j. For a node i, we defined in-neighbors which send information to node i as $\mathcal{N}_{\text{in}}^i = \{j \mid (j, i) \in \mathcal{E}\}$, and out-neighbors which receive information from node i as $\mathcal{N}_{\text{out}}^i = \{j \mid (i, j) \in \mathcal{E}$. We give the general assumptions for decentralized optimization [9].

Assumption 2.1: The communication network $\mathcal{G}$ is strongly connected.

Then we give some assumptions on objective functions.

Assumption 2.2: The entire objective function F is $\mu-$strongly convex for some constant $\mu > 0$, i.e.,

$$\nabla^2 F(x) \succeq \mu I_d, \forall x \in \mathbb{R}^d, \tag{2}$$

where μ is the strong convexity parameter.

Assumption 2.3: Each f_i is twice-differentiable, and both the gradient and Hessian are Lipschitz continuous, i.e.,

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L_1 \|x - y\|, \forall x, y \in \mathbb{R}^d, \tag{3}$$

and

$$\|\nabla^2 f_i(x) - \nabla^2 f_i(y)\| \leq L_2 \|x - y\|, \forall x, y \in \mathbb{R}^d, \tag{4}$$

where $L_1 \geq 0$ and $L_2 \geq 0$ are the Lipschitz constants of the local gradient and local Hessian, respectively.

Finally we give a common assumption on transmission delays.

Assumption 2.4: For any $(i, j) \in \mathcal{E}$, the transmission delay from any two nodes is bounded by a constant D satisfy $0 < D < \infty$.

B. Proposed Asynchronous Newton Tracking Mechanism

To model the process of sending messages between nodes in the network, we introduce an induced matrix $W \in \mathbb{R}^{n\times n}$ with entries w_{ij}. The matrix $W \in \mathbb{R}^{n\times n}$ is non-negative where $w_{ji} > 0$ if and only if node i can send information to node j, and the value of w_{ji} represents the weight assigned by node i to the information sent to node j. To estimate the global Hessian matrix, one common method is to use the dynamic average consensus method (DAC) [19], [23]. We first present DAC method under synchronization, showing clearly that this synchronous method cannot be directly applied to the asynchronous case. Then we propose our fully asynchronous Newton tracking mechanism. To this end, the synchronous DAC method is

$$H_i^{k+1}=H_i^k-\gamma\sum_{j\in\mathcal{N}_{\text{in}}^i} w_{ij}\left(H_i^k-H_j^k\right)+\nabla^2 f_i\left(x_i^{k+1}\right)-\nabla^2 f_i\left(x_i^k\right), \tag{5}$$

where H_i^k hold on node i is the estimation of the global Hessian matrix at time step k. Under synchronization, to estimate the global Hessian, we require that W is column stochastic, i.e., $\sum_{i=1}^n w_{ij} = 1, \forall j \in [n]$ [9], [24]. With this condition, by taking the average of both sides of (5) over all the nodes, we have

$$\frac{1}{n}\sum_{i=1}^n H_i^k = \frac{1}{n}\sum_{i=1}^n \nabla^2 f_i(x_i^k), \quad \forall k, \tag{6}$$

where we use the fact that W is column stochastic and the initialization $H_i^0 = \nabla^2 f_i(x_i^0)$, $\forall i \in [n]$. The property (6) is very important for each node to successfully track the global Hessian. However, in the asynchronous setting, node i may fail to receive H_j^k from node j at time step k due to the uncoordinated activations and staled information. This causes the property (6) to no longer hold, thus affecting the performance of the algorithm.

To tackle this issue, we use an augmented graph to store the delayed information from the neighbors before it is used [25], [26]. An example of an augmented graph is given in Fig.2, the left of the Fig.2 is the original graph including two nodes i and j. On the right is the augmented graph, where node j sends $\tilde{H}_j^k$ to node i at some time $t(k)$ and node i uses it at some time $t(k+3)$. Two matrices are corresponding induced matrices W and $\tilde{W}$ of the original graph and the augmented graph, respectively. Let $\hat{n} = n(D+1)$ be the number of nodes in the augmented graph, $\tilde{H}_i^k$ be the Hessian information on node $i \in [\hat{n}]$ at step k, and $\tilde{W} \in \mathbb{R}^{\hat{n}\times\hat{n}}$ be the induced matrix of the augmented graph. It has been shown that $\tilde{W} \in \mathbb{R}^{\hat{n}\times\hat{n}}$ is column stochastic [17]. Thus, for the augmented graph we have

$$\frac{1}{\hat{n}}\sum_{i=1}^{\hat{n}} \tilde{H}_i^k = \frac{1}{n}\sum_{i=1}^n \nabla^2 f(x_i^k). \tag{7}$$

This means that we still have the key property (7) in the asynchronous setting similar to (6) in the synchronous setting through the augmented graph. Note that the mixing of the Hessian information on node i, weighted by the i-th row of the induced matrix in the augmented graph, is equivalent to introducing a buffer to store $\frac{1}{|\mathcal{N}_{\text{out}}^j|}H_j$ and summing over the Hessian information in the buffer. For simplicity, we use a buffer to implement our asynchronous Newton tracking mechanism, as shown in Algorithm 1.

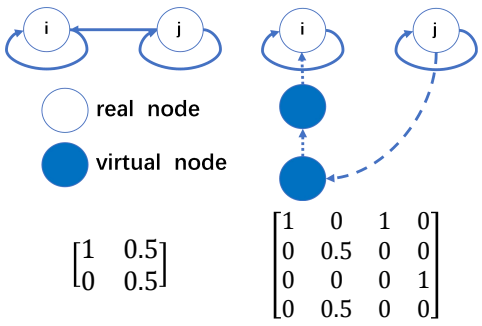


Fig. 2. Augmented graph and induced matrix

Algorithm 1 Proposed Asynchronous Newton Tracking Mechanism on node i

Initialize: $\gamma > 0$. $H_i^0 = \nabla^2 f_i(x_i^0)$. Create local buffers $\mathcal{H}_i$. Broadcast $\tilde{H}_i^0 := \frac{H_i^0}{|\mathcal{N}_{\text{out}}^i|}$ to out-neighbors.
Repeat
Keep receiving $\tilde{H}_j$ from $j \in \mathcal{N}_{\text{in}}^i$ and add it to the local buffer $\mathcal{H}_i$, until node i is activated.
if node i is activated **then**
$H_i^{k+1}=(1-\gamma)H_i^k+\gamma\textbf{sum}(\mathcal{H}_i)+\nabla^2 f_i(x_i^{k+1})-\nabla^2 f_i(x_i^k)$
Empty local buffer $\mathcal{H}_i$ and broadcast $\tilde{H}_i^{k+1} = \frac{H_i^{k+1}}{|\mathcal{N}_{\text{out}}^i|}$ to its out-neighbors.
end if
Output H_i^{k+1}

C. Implementations of Proposed Asynchronous Newton Tracking Mechanism

In this section, we combine the proposed mechanism with two existing asynchronous first-order algorithms, APPG [17] and ASY-SONATA [16], resulting in two asynchronous Newton algorithms which converge faster than the original APPG and SONATA. After that, we provide the communication-efficient implementation of the proposed Mechanism, which uses compression to avoid sending full matrices. Note that the existence of Hessian approximation in the algorithms makes convergence analysis challenging, and the analysis of APPG and SONATA is no longer applicable.

1) **APPG with Newton Tracking***:* The first algorithm is based on APPG [17], The first algorithm is based on APPG, which is an asynchronous robust algorithm developed from the push-pull algorithm [9]. APPG employs the DAC method to track the global gradient and use a buffer to store unused information that has been sent to reduce the impact of asynchrony on convergence. After each node obtains an approximation of the global gradient, we leverage our

Algorithm 2 APPG with Newton Tracking

Initialize: $\alpha > 0$, $M > 0$, x_i^0, $d_i^0 = y_i^0 = \nabla f_i(x_i^0)$, $H_i^0 = \nabla^2 f_i(x_i^0)$. Create local buffers $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{H}_i$. Broadcast $\tilde{x}_i^0 := x_i^0$, $\tilde{y}_i^0 := \frac{y_i^0}{|\mathcal{N}_{\text{out}}^i|}$, $\tilde{H}_i^0 := \frac{H_i^0}{|\mathcal{N}_{\text{out}}^i|}$ to out-neighbors

Repeat

Keep receiving $\tilde{x}_j, \tilde{y}_j, \tilde{H}_j$ from $j \in \mathcal{N}_{\text{in}}^i$ and add them to the local buffer $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{H}_i$, respectively.

if node i is activated **then**

$x_i^{k+1} = \textbf{avg}(\mathcal{X}_i) - \alpha d_i^k$

$y_i^{k+1} = \textbf{sum}(\mathcal{Y}_i) + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k)$

Apply algorithm 1 to get H_i^{k+1}

$d_i^{k+1} = (H_i^{k+1} + M\boldsymbol{I}_d)^{-1} y_i^{k+1}$

Empty buffer and broadcast $\tilde{x}_i^{k+1} = x_i^{k+1}$, $\tilde{y}_i^{k+1} = \frac{y_i^{k+1}}{|\mathcal{N}_{\text{out}}^i|}$, $\tilde{H}_i^{k+1} = \frac{H_i^{k+1}}{|\mathcal{N}_{\text{out}}^i|}$ to out-neighbors

end if

until stopping criterion

proposed mechanism to generate an approximation of the global Hessian, allowing each node to approximate the global Newton direction. This direction, with its inclusion of curvature information, is capable of accelerating convergence. The resulting algorithm is summarized in Algorithm 2.

2) **ASY-SONATA with Newton Tracking***:* We next combine the proposed asynchronous mechanism with ASY-SONATA [16] which presents a method for robustly estimating the global gradient asynchronously. In contrast to APPG that is developed from the push-pull algorithm, ASY-SANATA is based on the push-sum algorithm [6]. By integrating our Newton tracking mechanism, we enable each node to generate an approximation of the global Newton direction and get a second-order version of ASY-SONATA, as outlined in Algorithm 3.

Algorithm 3 Asy-SONATA with Newton Tracking

Initialize: $\alpha > 0$, $M > 0$, x_i^0, $d_i^0 = y_i^0 = \nabla f_i(x_i^0)$, $\rho_i^0 = 0$, $H_i^0 = \nabla^2 f_i(x_i^0)$. Create local buffers $\mathcal{X}_i$, $\mathcal{H}_i$, $\tilde{\boldsymbol{\rho}}_i$. Broadcast $\tilde{x}_i^0 := x_i^0$, ρ_i^0, $\tilde{H}_i^0 := \frac{H_i^0}{|\mathcal{N}_{\text{out}}^i|}$ to out-neighbors

Repeat

Keep receiving $\tilde{x}_j$, ρ_j, $\tilde{H}_j$ from $j \in \mathcal{N}_{\text{in}}^i$, replace buffer variables $\tilde{x}_{ij}$ with $\tilde{x}_j$, add $\tilde{H}_j$ to $\mathcal{H}_i$.

if node i is activated **then**

$x_i^{k+1} = x_i^k - \alpha d_i^k$

$x_i^{k+1} = \frac{1}{|\mathcal{N}_{\text{in}}^i|} x_i^{k+1} + \sum_{j \in \mathcal{N}_{\text{in}}^i} \frac{1}{|\mathcal{N}_{\text{in}}^i|} \tilde{x}_{ij}$

$\epsilon^{k+1} = \nabla f(x_i^{k+1})) - \nabla f(x_i^k))$

$y_i^{k+1} = \frac{1}{|\mathcal{N}_{\text{out}}^i|}(y_i^k + \sum_{j \in \mathcal{N}_{\text{in}}^i}(\rho_{ij} - \tilde{\rho}_{ij}) + \epsilon^{k+1}$

$\rho_i^{k+1} = \rho_i^k + y_i^{k+1}$

replace buffer variables $\tilde{\rho}_{ij}$ with ρ_j

Apply algorithm 1 to get H_i^{k+1}

$d_i^{k+1} = (H_i^{k+1} + M\boldsymbol{I}_d)^{-1} y_i^{k+1}$

Empty $\mathcal{H}_i$ and broadcast x_i^{k+1}, $\boldsymbol{\rho}_i^{k+1}$, $\tilde{H}_i^{k+1} := \frac{H_i^{k+1}}{|\mathcal{N}_{\text{out}}^i|}$ to out-neighbors

end if

until stopping criterion

III. Convergence Analysis

In this section, we provide a unified analysis for Algorithm 2 and Algorithm 3. For the sake of clarity, we provide the compact form over the augmented graph for both algorithms.

$$\begin{aligned} \widetilde{\boldsymbol{x}}^{k+1} &= \widetilde{A}^k(\widetilde{\boldsymbol{x}}^k - \alpha(\widetilde{\boldsymbol{H}}^k + MI)^{-1}\widetilde{\boldsymbol{y}}^k), \\ \widetilde{\boldsymbol{y}}^{k+1} &= \widetilde{B}^k\widetilde{\boldsymbol{y}}^k + \nabla(k+1) - \nabla(k), \end{aligned} \tag{8}$$

where

$$\begin{aligned} \widetilde{\boldsymbol{x}}^k &= [x^k; x^k(1); \cdots; x^k(D)] \in \mathbb{R}^{\hat{n}d} \\ \widetilde{\boldsymbol{y}}^k &= [y^k; y^k(1); \cdots; y^k(D)] \in \mathbb{R}^{\hat{n}d} \\ \widetilde{\boldsymbol{H}}^k &= [H^k; H^k(1); \cdots; H^k(D)] \in \mathbb{R}^{\hat{n}d \times d} \\ \nabla(k) &= [\nabla f(\tilde{\boldsymbol{x}}^k); \boldsymbol{0}_{(\hat{n}-n)d}], \end{aligned}$$

and $\widetilde{A}^k$ is a row-stochastic matrix and $\widetilde{B}^k$ is a column-stochastic matrix which are already defined in [17]. It is worth mentioning that $\widetilde{A}^k$ and $\widetilde{B}^k$ are time-varying matrices depending on the way of delay in each timestamp. Through the proof, we assume $d = 1$ (scale variables), which is quite standard in asynchronous optimization, see [16], [17].

Inspired by work [16], we provided the convergence analysis for (8) with Theorem 3.2 below under local condition, which establishes linear convergence for our mechanism. We leave global convergence analysis for further discussions.

Assumption 3.1: For local convergence analysis, we assume each node is close to optimal point $\mathbf{x}^*$. Specifically, we have (9) for $k \geq N_1$.

$$\|H_i^k - H^*\|_F \leq r, \forall i \in [n], \tag{9}$$

where H^* is the hessian matrix of the optimal point $\mathbf{x}^*$.

Theorem 3.2: Let $\mathbf{x}^*$ denote the unique optimal solution of (1) and $\left\{\left(\mathbf{x}_i^k\right)_{i=1}^n\right\}_k$ be the sequence generated by (8). With prior assumptions hold, there exists a constant $\bar{\alpha} < 1$ such that when $\alpha \leq \bar{\alpha}$, the asynchronous algorithms holds

$$\left\|\mathbf{x}^k - \mathbf{1}_n \otimes \mathbf{x}^\star\right\| = \mathcal{O}\left(\lambda^k\right) \tag{10}$$

where $\lambda \in (0, 1)$ is given by (20).

Remark 3.3: By properly choosing the stepsize, we can show that our decay factor λ is better than the factor of first-order methods derived in [16]. It indicates that our algorithm enjoys a sharper local linear convergence rate.

Proof: Build upon on prior idea in [16], we take the advantage of the Hessian approximation's curvature information to further bound the error quantities. Specifically, we utilize the locally quadratic convergence rate of the centralized Newton's method to bound the optimization error.

A. Step I

We adapt a unified model to study the dynamics of the consensus and optimization errors of the asynchronous framework, which consists in pulling out the tracking update and treating the variables term $-\alpha(\widetilde{H}^k + MI)^{-1}\widetilde{y}^k$ as an exogenous perturbation δ^k. Our framework has the following scheme in compact form as

$$\widetilde{\boldsymbol{x}}^{k+1} = \widetilde{A}^k(\widetilde{\boldsymbol{x}}^k + \delta^k), \tag{11}$$

Applying (11) recursively, we have

$$\widetilde{\boldsymbol{x}}^{k+1} = \widetilde{A}^{k:0}\widetilde{\boldsymbol{x}}^{0} + \sum_{l=0}^{k} \widetilde{A}^{k:l}\delta^{l}. \tag{12}$$

where $\widetilde{A}^{k:0} \triangleq \widetilde{A}^{k}\widetilde{A}^{k-1}\dots\widetilde{A}^{0}$. Based on Proposition 18 in [16], we have

$$\|\widetilde{\boldsymbol{x}}^{k+1} - \mathbf{1}x_{\psi}^{k+1}\| \leq C_2\rho^{k}\|\widetilde{\boldsymbol{x}}^{0} - \mathbf{1}x_{\psi}^{0}\| + C_2\sum_{l=0}^{k}\rho^{k-l}|\delta^{l}|, \tag{13}$$

where x_{ψ}^{k+1} satisfies $x_{\psi}^{k+1} = x_{\psi}^{k} + \psi_i^{k}\delta^{k}$ for a sequence of stochastic vectors $\{\psi^k\}$ and $\rho \in (0,1), C_2 > 0$ are given in [16, Lemma 17].

B. Step II

In this subsection, we focus on consensus agreement, newton direction tracking error and optimization error for our framework, the consensus error and optimization error at iteration k are defined as

$$E_c^k \triangleq \|\widetilde{\boldsymbol{x}}^{k} - \mathbf{1}x_{\psi}^{k}|, \quad E_o^k \triangleq |x_{\psi}^{k} - x^*|.$$

Similar to the definition of Gradient Tracking Error in [16], we give the gradient tracking error and newton direction tracking error at iteration k along with the magnitude of the tracking variables are defined as

$$E_{\mathbf{t}}^k \triangleq \left|y_i^k - \xi_i^{k-1}\bar{g}^k\right|, E_{\mathbf{y}}^k \triangleq \left|y_i^k\right|, \bar{g}^k \triangleq \frac{1}{n}\sum_{i=1}^{n}\nabla f_i\left(x_i^k\right)$$
$$E_d^k \triangleq |d_i^k|, E_{dt}^k \triangleq |d_i^k - \xi_i^{k-1}\bar{d}^k|,$$

where $\bar{d}^k = \frac{1}{n}\sum_{i=1}^{n}(\nabla^2 f_i(x_i))^{-1}\nabla f_i(x_i)$, $\{\xi_k\}$ is the sequence of stochastic vector introduced in [16, Lemma 15]. Next, we build the connection between these error quantities.

Proposition 3.4: The error quantities satisfy: for all $k \in \mathcal{N}_0$, there exists some positive constants C_0, C_2, such that

$$E_c^{k+1} \leq C_2\rho E_c^0 + C_2\sum_{l=0}^{k}\rho^{k-l}\alpha^{l}E_d^{l} \tag{14}$$

$$E_t^{k+1} \leq C_0\rho^{k}\|\mathbf{g}^0\| + 3C_0L_1\sum_{l=0}^{k}\rho^{k-l}(E_c^{l} + \alpha^{l}E_d^{l}) \tag{15}$$

$$E_d^{k} \leq \frac{1}{M_1}E_t^{k} + \frac{L_1}{\mu\sqrt{n}}E_c^{k} + E_o^{k} \tag{16}$$

$$E_{dt}^{k} \leq \frac{1}{M_1}E_t^{k} \tag{17}$$

$$E_{\mathbf{o}}^{k+1} \leq \sum_{l=0}^{k}\left(\prod_{t=l+1}^{k}\left(1-\eta^2\alpha+\eta^2\alpha\delta^{t}\right)\right)\left(\frac{L_1}{\mu\sqrt{n}}E_{\mathbf{c}}^{l}+E_{\mathbf{dt}}^{l}\right)\alpha^{l} + \prod_{t=0}^{k}\left(1-\eta^2\alpha+\eta^2\alpha\delta^{t}\right)E_{\mathbf{o}}^{0} \tag{18}$$

C. Step III

Based on [16, Theorem 23] and Proposition 3.4, we can build the linear dynamics system as (21), where $|E_{\mathbf{d}}|^{\lambda,N}, |E_{\mathbf{c}}|^{\lambda,N}, |E_{\mathbf{t}}|^{\lambda,N}, |E_{\mathbf{o}}|^{\lambda,N}$ are defined based on [16, Lemma 21]. By [16, Theorem 23], if $\rho(K) < 1$, the algorithm vanish with linear convergence rate. According to Lemma 24 in [16], $\rho(K) < 1$ if and only if $p_K(1) > 0$. The characteristic polynomial of $\mathbf{K}$ is

$$x^4 - \frac{\mathrm{b_1C_2}x^2\alpha+\mathrm{b_1C_2}x\alpha^2(\lambda-\mathcal{L}(\alpha))}{(\lambda-\rho)(\lambda-\mathcal{L}(\alpha))} - \frac{\mathrm{b_2C_2}x\alpha+b_2x^2\alpha(\lambda-\rho)}{M_1(\lambda-\rho)^2} - \frac{\mathrm{b_2C_2}\alpha^2}{M_1(\lambda-\rho)^2(\lambda-\mathcal{L}(\alpha))} - \frac{\mathrm{b_2}x\alpha^2}{M_1(\lambda-\rho)(\lambda-\mathcal{L}(\alpha))} \tag{19}$$

Consider the the continuity of equation (19) and $p_K(1) > 0$, let $x = 1$, $\lambda = 1$, we obtain $\bar{\alpha}_1 = \frac{1}{J_1}$, where

$$J_1 = \frac{b_1C_2M_1(1-\rho)+b_2C_2+b_2(1-\rho)}{M_1(1-\rho)^2} + \frac{b_1C_2}{(1-\rho)(\eta^2-\eta^2\delta)} + \frac{b_2C_2}{M_1(1-\rho)^2(\eta^2-\eta^2\delta)} + \frac{b_2}{M_1(1-\rho)(\eta^2-\eta^2\delta)}$$

For term $\frac{b_1C_2}{(1-\rho)\eta^2(1-\delta)} \geq b_1C_2 \geq \frac{L_1}{\mu\sqrt{n}}2\sqrt{(D+2)n} = \frac{L_1}{\mu}2\sqrt{D+2} > 1$, it's easy to check other terms are all positive, so we have $\bar{\alpha}_1 < 1$. Therefore, when $0 < \alpha < \bar{\alpha}_1$, we have

$$E_c^k = \mathcal{O}(\lambda^k), E_d^k = \mathcal{O}(\lambda^k), E_t^k = \mathcal{O}(\lambda^k), E_o^k = \mathcal{O}(\lambda^k)$$

with $\lambda \in (0,1)$ given by

$$\lambda = \max(\rho+\sqrt{(b_1C_2+\frac{b_2C_2}{M_1}+\frac{b_2}{M_1})(1+\frac{1}{\epsilon})\alpha}, 1-\alpha(\eta^2-\eta^2\delta-\epsilon)), \tag{20}$$

where $\delta > 0$ is arbitrary small, $\epsilon > 0$ is properly chosen. ■

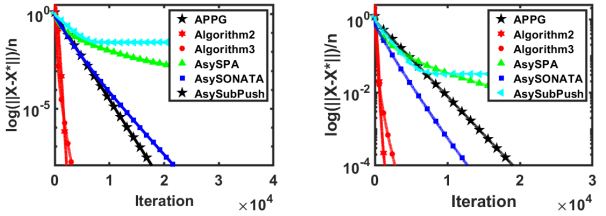
IV. Numerical Experiments

In this section, we numerically test our theoretical findings and compare with existing asynchronous methods and synchronous newton tracking method in the decentralized case over two classes of problems. The network is generated randomly, firstly generate a cycle graph which guarantee strong connected of the graph, then given specific number of out-neighbors $\mathcal{N}_{\text{out}} \leq n$, we add edges randomly. The optimal point of the problem x^* will be precomputed by centralized newton method.

The asynchronous model we adapt to test iteration performance is common in asynchronous optimization, see [16], [27]. In fully asynchronous setting, a virtual global clock that different from [20] which need real global clock for coordination is usually assumed to record the whole iteration of the network, no matter which node complete a update, k would increase one. The agents follow the generated activation list and transmit information to their out-neighbors immediately after complete their own local update. The information from node i will not be available to j after the virtual global iteration $k + T_{ij}^k$. Transmitted delay T_{ij}^k is modeled by (integer) traveling time which is sampled

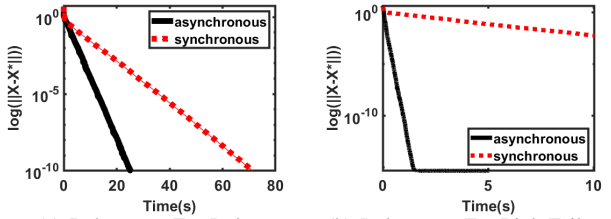
$$\begin{bmatrix} |E_{\mathbf{d}}|^{\lambda,N} \\ |E_{\mathbf{c}}|^{\lambda,N} \\ |E_{\mathbf{t}}|^{\lambda,N} \\ |E_{\mathbf{o}}|^{\lambda,N} \end{bmatrix} \preccurlyeq \underbrace{\begin{bmatrix} 0 & b_1 & \frac{1}{M_1} & 1 \\ \frac{C_2\alpha}{\lambda-\rho} & 0 & 0 & 0 \\ \frac{b_2\alpha}{\lambda-\rho} & \frac{b_2}{\lambda-\rho} & 0 & 0 \\ 0 & \frac{b_1\alpha}{\lambda-\mathcal{L}(\alpha)} & \frac{\alpha}{M_1(\lambda-\mathcal{L}(\alpha))} & 0 \end{bmatrix}}_{\triangleq \mathbf{K}} \begin{bmatrix} |E_{\mathbf{d}}|^{\lambda,N} \\ |E_{\mathbf{c}}|^{\lambda,N} \\ |E_{\mathbf{t}}|^{\lambda,N} \\ |E_{\mathbf{o}}|^{\lambda,N} \end{bmatrix} + \mathbf{c}, \tag{21}$$

where $b_1 \triangleq \frac{L_1}{\mu\sqrt{n}}, b_2 \triangleq 3C_0L_1, \mathcal{L}(\alpha) = 1 - \eta^2\alpha + \eta^2\alpha\delta$.



(a) MaxDelay = 10 (b) MaxDelay = 40

Fig. 3. Iteration Performance for Least Square



(a) Robustness For Delay (b) Robustness For Link Failure

Fig. 4. Wall-Clock Time Performance for Least Square

uniformly from the interval $[0, D_i^{\max}]$, where $D_i^{\max}$ is the maximum delay. We implement this asynchronous model in MATLAB on a Windows workstation with 2.20GHz 12-core Intel i7-8700 CPU. For experiments associated with wall-clock time, we adapt the Message Passing Interface (MPI) which already packed in Python as mpi4py [28] to simulate a network with multiple connected nodes [15], [18], which is implemented in 12-core Ubuntu 20.04.

A. Least Square

Least Square is an typical instance of (1), where $f_i = \|A_ix_i - b_i\|^2$, A_i is the feature matrix and b_i is the label vector privately known by the agent i. The whole data A and b are randomly generated from standard Guassian distribution and normalize it by its spectral norm, then we partition it into n parts privately stored in local agents.

1) Comparison with Asynchronous First Order Methods: We compare the asynchronous newton tracking mechanism Algorithm 2 and Algorithm 3 with all existing asynchronous methods for iteration performance. Figure 3(a) and 3(b) show the iteration performance for 30 agents with 8 out_neighbors in the asynchronous model, where *MaxTravelTime* is 10 and 40 respectively. We manually adjust the step-size in order to get best performance of each algorithm. From figure we observe that methods belong to the same kind mechanism share the similar feature, and our mechanism efficiently accelerate the asynchronous first-order algorithms. Additionally, in more asynchronous scenarios, we observe that these first order algorithms especially APPG suffers a lot if the network become more asynchronous, while our mechanism are more robust.

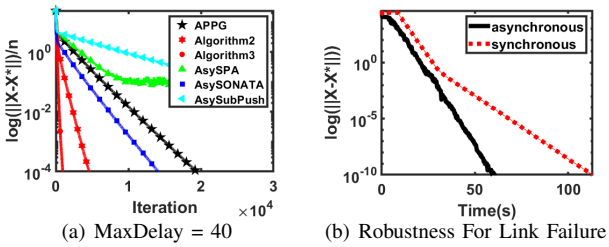
2) Comparison with Synchronous Newton Tracking: We compare the robustness of our mechanism with synchronous newton tracking from two perspectives, link failure and transmission delay. Synchronous newton tracking adapts synchronous peer to peer communication in mpi4py, while asynchronous method uses asynchronous peer to peer communication which needs *Attach_buffer()* initially. Fig 4(a) shows the running time performance for 10 agents with different link failure probability $0.01(1+\frac{i}{4})$. Fig 4(b) shows the running time performance for 10 agents with different delays $\text{delay}_i = 0.05 * (1 + \frac{i}{5})$, while each node keeps 100 observations, the dimension of features is 20. In addition, the iterations of asynchronous algorithms for agents range from 177 to 472 and the iterations of synchronous ones runs about 221 iterations. Results show our asynchronous newton tracking methods is more robust to the link failure, delay, and heterogeneous nodes, especially the slow nodes.

B. Logistic Regression

To further confirm the efficiency of the proposed mechanism, we solved the Two-Class logistic regression problem as follows.

$$x^* = \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} \frac{\rho}{2}\|x\|^2 + \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln\left(1 + \exp\left(-\left(\mathbf{o}_{ij}^T x\right)\mathbf{p}_{ij}\right)\right),$$

where each node i privately owns m_i training samples $(\mathbf{o}_{ij}, \mathbf{p}_{ij}) \in \mathbb{R}^d \times \{-1, +1\}, j = 1, \ldots, m$. The elements of $\mathbf{o}_{ij}$ are randomly generated following the standard Gaussian distribution and those of $\mathbf{p}_{ij}$ are generated following the uniform distribution on $\{-1, 1\}$, $\frac{\rho}{2}\|x\|^2$ is a regularization term to avoid over-fitting. The network setting is the same as ridge regression, figure 5(a), and figure 5(b) collaborate the results in the least square. In asynchronous iteration performance comparison, we tested iteration performance over synthetic data, and for wall-clock time comparison with synchronous newton tracking, we tested it over real dataset, Covtype. Each agent keeps 1000 observations with different delays $\text{delay}_i = 0.1 * (1 + \frac{i}{5})$, the dimension of Covtype is 54.



(a) MaxDelay = 40 (b) Robustness For Link Failure

Fig. 5. Numeric Experiments For Logistic Regression

V. Conclusions

In this paper, we have proposed a novel asynchronous newton tracking mechanism, which explicit second-order information under the fully asynchronous setting. Theoretical and empirical results demonstrate the efficiency and robustness of our mechanism compared with existing asynchronous methods. We leave several interesting directions for future work. In theoretical part, local condition is assumed, while it is possible to relax this condition and get a global convergence analysis. The other interesting direction is to introduce multi-consensus mechanism for our asynchronous newton tracking, there lies interesting balance between computation and communication for the fully asynchronous decentralized optimization.

References

[1] M. Liu, W. Zhang, Y. Mroueh, X. Cui, J. Ross, T. Yang, and P. Das, "A decentralized parallel algorithm for training generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 056–11 070, 2020.

[2] A. Beznosikov, G. Scutari, A. Rogozin, and A. Gasnikov, "Distributed saddle-point problems under data similarity," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8172–8184, 2021.

[3] J. Qin, J. Wang, L. Shi, and Y. Kang, "Randomized consensus-based distributed kalman filtering over wireless sensor networks," *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3794–3801, 2020.

[4] P. Dvurechenskii, D. Dvinskikh, A. Gasnikov, C. Uribe, and A. Nedich, "Decentralize and randomize: Faster algorithm for wasserstein barycenters," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[5] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[6] C. Xi, V. S. Mai, R. Xin, E. H. Abed, and U. A. Khan, "Linear convergence in optimization over directed graphs with row-stochastic matrices," *IEEE Transactions on Automatic Control*, vol. 63, no. 10, pp. 3558–3565, 2018.

[7] H. Liu, J. Zhang, A. M.-C. So, and Q. Ling, "A communication-efficient decentralized newton's method with provably faster convergence," *arXiv preprint arXiv:2210.00184*, 2022.

[8] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 601–615, 2014.

[9] S. Pu, W. Shi, J. Xu, and A. Nedić, "Push–pull gradient methods for distributed optimization in networks," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 1–16, 2020.

[10] S. Kumar, R. Jain, and K. Rajawat, "Asynchronous optimization over heterogeneous networks via consensus admm," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 1, pp. 114–129, 2016.

[11] M. Eisen, A. Mokhtari, and A. Ribeiro, "Decentralized quasi-newton methods," *IEEE Transactions on Signal Processing*, vol. 65, no. 10, pp. 2613–2628, 2017.

[12] F. Mansoori and E. Wei, "A fast distributed asynchronous newton-based optimization algorithm," *IEEE Transactions on Automatic Control*, vol. 65, no. 7, pp. 2769–2784, 2019.

[13] P. Lin, W. Ren, and Y. Song, "Distributed multi-agent optimization subject to nonidentical constraints and communication delays," *Automatica*, vol. 65, pp. 120–131, 2016.

[14] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Convergence of asynchronous distributed gradient methods over stochastic networks," *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 434–448, 2017.

[15] M. S. Assran and M. G. Rabbat, "Asynchronous gradient push," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 168–183, 2020.

[16] Y. Tian, Y. Sun, and G. Scutari, "Achieving linear convergence in distributed asynchronous multiagent optimization," *IEEE Transactions on Automatic Control*, vol. 65, no. 12, pp. 5264–5279, 2020.

[17] J. Zhang and K. You, "Fully asynchronous distributed optimization with linear convergence in directed networks," *arXiv preprint arXiv:1901.08215*, 2019.

[18] ——, "Asyspa: An exact asynchronous algorithm for convex optimization over digraphs," *IEEE Transactions on Automatic Control*, vol. 65, no. 6, pp. 2494–2509, 2019.

[19] G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 3, pp. 1245–1260, 2017.

[20] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* IEEE, 2003, pp. 482–491.

[21] N. Bof, R. Carli, G. Notarstefano, L. Schenato, and D. Varagnolo, "Multiagent newton–raphson optimization over lossy networks," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2983–2990, 2018.

[22] F. Mansoori and E. Wei, "Superlinearly convergent asynchronous distributed network newton method," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2874–2879.

[23] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes," in *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE, 2015, pp. 2055–2060.

[24] S. Pu, "A robust gradient tracking method for distributed optimization over directed networks," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 2335–2341.

[25] M. Cao, A. S. Morse, and B. D. Anderson, "Reaching a consensus in a dynamically changing environment: A graphical approach," *SIAM Journal on Control and Optimization*, vol. 47, no. 2, pp. 575–600, 2008.

[26] C. N. Hadjicostis and T. Charalambous, "Average consensus in the presence of delays in directed graph topologies," *IEEE Transactions on Automatic Control*, vol. 59, no. 3, pp. 763–768, 2013.

[27] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[28] L. Dalcín, R. Paz, and M. Storti, "Mpi for python," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1108–1115, 2005.