A Flexible Distributed Stochastic Optimization Framework for Concurrent Tasks in Processing Networks

Zai Shi, Atilla Eryilmaz, *Senior Member, IEEE*

***Abstract*—Distributed stochastic optimization has important applications in the practical implementation of machine learning and signal processing setup by providing means to allow interconnected network of processors to work towards the optimization of a global objective with intermittent communication. Existing works on distributed stochastic optimization predominantly assume all the processors storing related data to perform updates for the optimization task in each iteration. However, such optimization processes are typically executed at shared computing/data centers along with other concurrent tasks. Therefore, it is necessary to develop efficient optimization methods that possess the flexibility to share the computing resources with other ongoing tasks. In this work, we propose a new first-order framework that allows this flexibility through a probabilistic computing resource allocation strategy while guaranteeing the satisfactory performance of distributed stochastic optimization. Our results, both analytical and numerical, show that by controlling a flexibility parameter, our suite of algorithms (designed for various scenarios) can achieve the lower computation and communication costs of distributed stochastic optimization than their inflexible counterparts. This framework also enables the fair sharing of the common resources with other concurrent tasks being processed by the processing network.**

I. Introduction

Distributed stochastic optimization concerns the minimization of the average $\frac{1}{n}\sum_{i=1}^{n} f_i(x)$ of functions f_i in the form of $\mathbb{E}_{\xi_i\sim D_i} F_i(x;\xi_i)$, where D_i is the distribution of random variable ξ_i. Each f_i or its realizations are only accessible by a unique processor in a network, where communication can only happen between the neighbors to find the optimum of the above global objective. This setup has found many applications in machine learning and signal processing fields. For example, when faced with a big dataset, we often divide it to several small datasets and process them in different servers connected by a network, such as MapReduce scheme [1]. If the problem is empirical risk minimization [2] for supervised learning, then f_i represents the loss function of the local dataset in Server i and F_i is the loss function associated with its single data point. Each processor needs to determine the parameter of the prediction function that minimizes the average loss over the entire dataset. This process can be formulated as distributed optimization. Another example is decentralized estimation. In a wireless sensor network, each sensor has a measurement of the parameter that we are interested in. Each measurement contains noise from the environment following a certain distribution, maybe different from other sensors. How to estimate the parameter based on all the measurements is also a distributed stochastic optimization problem.

One of the pioneering works on distributed optimization was Tsitsiklis et al.'s work [3]. Since then, several types of methods have been proposed for deterministic optimization, such as distributed subgradient descent (DSD) [4], [5], distributed dual averaging [6], [7], alternating direction method of multipliers (ADMM) [8], [9], Nesterov's method [10], [11] and second-order algorithm [12], [13], with different performances and restrictions. In parallel, a lot of works were focusing on their stochastic optimization counterpart, which will be summarized in Section I-A. Among these types, DSD and its stochastic counterpart Distributed Stochastic Subgradient Algorithm (DSSA) [14] are the most important algorithms because they are easily implemented in a distributed way (ADMM needs sequential variable updates and second order methods need costly distributed Hessian calculation), and the basis of many further developed algorithms. So in this paper, we will explore first-order methods built on these fundamental methods.

Meanwhile, we notice that prior works predominantly consider the scenario of *simultaneous updates of all network processors* for single distributed optimization task. However, in real processing networks, multiple tasks are typically performed simultaneously by the same computing resources. Consequently, the computing resources must necessarily be shared amongst concurrent tasks including distributed optimization for satisfactory performance, which is largely overlooked in prior works.

This motivates us in this work to propose a set of flexible distributed stochastic subgradient algorithms that allow processors to work on multiple ongoing tasks simultaneously by probabilistically switching between them. Our contributions can be summarized as follows:

- We introduce a randomization parameter into DSSA that allows sharing of resources amongst concurrent tasks being processed by the network. This mechanism provides the necessary flexibility without disturbing the distributed nature of the optimization. The algorithm is called Partially Updated Stochastic Subgradient Descent (PUSSD) and its details will be presented in Section II

This work was supported in part by NSF grants: CNS-NeTS-1717045, CMMI-SMOR-1562065, CNS-ICN-WEN-1719371, CNS-SpecEES-1824337, CNS-NeTS-2007231; the ONR Grant N00014-19-1-2621; the DTRA grant: HDTRA1-18-1-0050.

Zai Shi and Atilla Eryilmaz are with Department of Electrical and Computer Engineering, the Ohio State University, Columbus, 43210, US (email: shi.960@osu.edu; eryilmaz.2@osu.edu).

- We analyze the performance of our randomized distributed stochastic optimization framework for various classes of functions. Our investigations show that for various classes of functions, the convergence and convergence rate characteristics of the traditional DSSA [14] are maintained while parallel processing ability can be achieved with lower computation costs in our framework. The discussion will be presented in Section III.
- Moreover, we develop a variant of our basic algorithm to further reduce the communication cost of our basic algorithm with less communication overhead in situations where communication costs are non-negligible with respect to computation costs. This variant will be introduced in Section IV.
- Through the experiments of three types of regression problems in Section V, the efficiency and the fairness of our algorithms are clearly demonstrated when compared with traditional and newly-proposed schemes.

A. Related Works

In this subsection, we introduce some related works on distributed stochastic optimization.

As mentioned above, DSSA proposed by Ram et al. [14] is the most classical method for distributed stochastic optimization. They presented different convergence results based on different stepsizes and stochastic errors. After that several methods were proposed with certain convergence rates, which is measured by optimality gap after T iterations of the algorithm. In [6], Duchi et al. showed that their distributed dual averaging (DDA) algorithm can use stochastic gradients to achieve a $O(n\log T/\sqrt{T})$ rate for general convex functions. For time-varying directed graphs with strongly convex objectives, A subgradient-push method [15] was proposed by Nedic and Olshevsky with a convergence rate of $O(\log T/T)$. A class of decentralized primal-dual type algorithms were presented in [16], which possesses a $O(1/\sqrt{T})$ (respectively, $O(1/T)$) convergence rate for general convex (respectively, strongly convex) functions. For nonconvex functions, a rate of $O(1/T+1/\sqrt{nT})$ can be achieved in [17] to obtain a first-order stationary solution. To get a faster convergence for strongly convex functions, the work [18] used a time-dependent weighted mixing of history values as the final output and proved its $O(n\sqrt{n}/T)$ rate. For random networks, the authors in [19] presented a method with a $O(1/T)$ rate when the objective is strongly convex. Two methods called DSGT and GSGT were proposed in [20] for different purposes, and both of them can achieve a linear rate converging to a neighborhood of the optimality and a $O(1/T)$ rate converging to the exact solution for smooth and strongly convex functions.

Note that the above methods all require simultaneous updates of all processors in the processing network.

Meanwhile, there is a newly-proposed optimization scheme called Sparsified Stochastic Gradient Descent (SGD) [21]–[26] for distributed machine learning, which shares some similarities with our method. However, our method is fundamentally different from this scheme. First of all, the motivation of sparsified SGD is to reduce commnication cost by only communicating top k components of gradients, while our method is designed to enable flexible resource allocation for concurrent tasks in processing networks, which can potentially reduce computation cost. Secondly, our method does not have a procedure called error compensations, which is critical for the convergence results of sparisified SGD. It is an advantage of our method because we do not need additional memory to store errors. Thirdly, our method is a truly decentralized method without any master node collecting all the information from other nodes, while it is not true for sparsified SGD.

B. Notation.

Throughout the rest of the paper, n is the number of processors in the network. In the algorithms, $x_i^k, u_i^k, y_i^k, z_i^k$ are the values of Processor i in iteration k and a_{ij}^k is the weight which processor i puts on the value sent from Processor j. $\partial f(x)$ refers to the set of subgradients of the function f at x and $\nabla f(x)$ is the gradient of f at x. $||\centerdot||_p$ denotes the l_p-norm for vectors. If A is a matrix, then A_{ij} refers to the (i,j)th entry of A. A^T is defined as the transpose of A. $\pi_\chi\{\centerdot\}$ is the projection operator defined as

$$\pi_\chi\{x\} = \arg\min_{y\in\mathcal{X}} ||y-x||_2^2.$$

The complexity of projection is dependent on the structure of χ and it has a closed form for a simplex or hypercube set.

II. Model and Basic Algorithm

We consider the following stochastic optimization problem processed by a network consisting of n processors:

$$\min_{x\in\chi} f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{1}$$

where $f_i(x)$ is convex and in the form of $\mathbb{E}_{\xi_i\sim D_i}F_i(x;\xi_i)$ for all i, and $\chi\subseteq\mathbb{R}^d$ is a convex set. D_i is a distribution which may be different from other i's. The network is represented by a static connected undirected graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$. Processor i can only have access to its own local function f_i and compute its stochastic (sub)gradients $\tilde{g}_i(x)\in\partial F_i(x;\xi_i(t))$ for a sample $\xi_i(t)$, whose expectation is a subgradient of $f_i(x)$. Meanwhile, Processor i can communicate its values with its neighbors, the set of which is denoted as $N(i)$. We assume that the computation cost of computing a stochastic subgradient is 1 and the communication cost of one link between two processors is τ, which can be larger or smaller than 1.

In Section I, we already mentioned the application of distributed stochastic optimization. Here we take one example to explain why we are interested in solving (1) within our model. In supervised learning setup, we consider the situation where the training dataset is stored separately in several servers interconnected by a network. If each server wants to solve the empirical risk minimization problem based on its stored training dataset, then we can write $f_i(\theta)=\sum_{j=1}^{n_i} l(h(x_{i,j};\theta),y_{i,j})$ where $h(\centerdot)$ is the prediction function such as a linear model [2], $l(\centerdot)$ is the loss function such as l_2 norm loss [2], $(x_{i,j},y_{i,j})$

is the jth training data point of the total n_i data points stored in Server i and θ is the global variable that the servers collaborate to optimize. Obviously Processor i can compute a stochastic gradient $\nabla_\theta l(h(x_{i,m};\theta), y_{i,m})$ of its local loss function by randomly choosing a data point $(x_{i,m}, y_{i,m})$ from the local dataset. Stochastic gradients are preferred to be used in machine learning practice [2] because the computation cost of a stochastic gradient is much less than a true gradient when the dataset is large. The details of this example will be shown in Section V.

We assume that (1) has an nonempty optimality set S and denote $x^* \in S$. Meanwhile, we give the following assumptions about $f_i(x)$, some of which may be used in the next section:

Assumption 1. *$f_i(x)$ has bounded stochastic subgradients for any i and $x \in \chi$, i.e., $||\tilde{g}_i||_2 \leq C$ for some constant C, where $\tilde{g}_i$ is a stochastic subgradient of f_i.*

Assumption 2. *$f_i(x)$ is L-smooth for any i and $x \in \chi$, i.e., $||\nabla f_i(x) - \nabla f_i(y)||_2 \leq L||x-y||_2$, where $x, y \in \chi$.*

Assumption 3. *$f_i(x)$ is μ-strongly convex for any i and $x \in \chi$, i.e., $f_i(y) \geq f_i(x) + g_i^T(y-x) + \frac{\mu}{2}||y-x||_2^2$, where $g_i \in \partial f_i(x)$ and $x, y \in \chi$.*

It is noted that when Assumption 1 is assumed, we have

$$f(y) - f(x) \leq C||y-x||_2$$

for any $x, y \in \chi$ by the definition of subgradients. Meanwhile, the above assumptions are only needed to be satisfied when $x \in \chi$.

The goal of distributed optimization is to make the value x_i^k of each processor converge to the optimal solution of (1) using limited communication with neighbors. Previous methods predominantly focus on solving this problem where each processor updates its value concurrently in each iteration. One of the questions we address in this work is whether it is possible to make a subset of the processors update the values in each iteration while still guaranteeing the convergence of the algorithm. Such a partial update may potentially reduce the computation cost of the network by providing flexibility if each processor is doing multi-tasks. When an processor is not involved in the process of solving (1) in this iteration, it can spare its resource in other tasks, such as measurement, data preprocessing or other distributed optimization problems.

Inspired by the above motivation, we propose Partially Updated Stochastic Subradient Descent (PUSSD) shown in Algorithm 1, which can be implemented distributedly in each processor. The dynamics of Algorithm 1 are as follows:

$$x_i^k = a_{ii}^k u_i^{k-1} + \sum_{j \in N(i)} a_{ij}^k u_j^{k-1}$$

$$u_i^k = \begin{cases} \pi_\chi\{x_i^k - \eta_k \tilde{g}_i^k\} & \text{if } r < p \text{ for processor } i \\ x_i^k & \text{else} \end{cases}$$

This algorithm can be divided into two operations. Steps 6 and 7 refer to the communication operation, where each processor communicates with its neighbors to average the information from its own and the neighbors. Steps 8 to 14 refer to the computation operation, where a subset of processors compute the subgradients and run the subgradient descent to approach the optimal solution of (1). The other processors just maintain their original values and process different tasks. Using this algorithm, we can process several tasks concurrently in this network with different emphasis on the distributed optimization task by choosing p values. The traditional DSSA [14] corresponds to our algorithm when $p = 1$. In this way, PUSSD can be regarded as a generalization of the traditional DSSA by providing the flexible parallel processing ability.

Algorithm 1 Partially Updated Stochastic Subgradient Descent (PUSSD) for Processor i

1: **Input:** number of iterations K, probability to run subgradient descent p, stepsizes for K iterations $\{\eta_k\}_{k=1}^K$.
2: **Output:** Option I: x_i^K; Option II: $v_i^K := \frac{\sum_{k=1}^K \eta_k x_i^k}{\sum_{k=1}^K \eta_k}$; Option III: $z_i^K := \frac{2}{K(K+1)} \sum_{k=1}^K k x_i^k$
3: Initialize $x_i^0 \in \chi$.
4: Set $u_i^0 \leftarrow x_i^0$.
5: **for** $k = 1$ **to** K **do**
6: Send u_i^{k-1} to the neighbor processors and receive $\{u_j^{k-1}\}_{j \in N(i)}$ from all the neighbor processors.
7: Set $x_i^k \leftarrow a_{ii}^k u_i^{k-1} + \sum_{j \in N(i)} a_{ij}^k u_j^{k-1}$, where a_{ii}^k, a_{ij}^k satisfy Assumption 4.
8: Generate a random number r in $[0, 1]$.
9: **if** $r < p$ **then**
10: Compute a stochastic subgradient $\tilde{g}_i^k$ of $f_i(x_i^k)$.
11: Set $u_i^k \leftarrow \pi_\chi\{x_i^k - \eta_k \tilde{g}_i^k\}$.
12: **else**
13: Set $u_i^k \leftarrow x_i^k$ and process other tasks.
14: **end if**
15: **end for**

Meanwhile, there exists some hard-partitioning schemes, such as assigning a partition of processors to a certain task, which also allows for parallel processing. But we should mention that the distributed optimization framework is especially important under many circumstances whereby different processors have access to chunks of data that may be private or infeasible to share with other processors in the network. The example of wireless sensor networks in Section I is such a case where each measurement is related to its location and not easy to be shared. So instead we prefer to let each processor share its updated parameters with its neighbors. Hard-partitioning schemes over network processors may be difficult to realize this end, especially under the above-mentioned circumstances. Meanwhile, the aim of the probabilistic switching is to make the gradient computed in each step a good estimation of the full gradient of the global function, which is essential for the convergence results shown in the next section. Other alternatives such as time division multiplexing can be regarded as specific implementations of sequential processing, which has been compared with our methods in Section V.

In the next section, we will show how the probabilistic

switching impacts convergence results of our algorithm.

III. Convergence Results for PUSSD

First we give some definitions prepared for an assumption related to the communication operation of PUSSD. Define $A(k) \in \mathbb{R}^{n\times n}$ as the weight matrix in iteration k whose (i,j)th entry is a_{ij}^k. Because of the communication limitation, $a_{ij}^k \neq 0$ only when edge $(i,j) \in \mathcal{E}$ or $j = i$. Supposing $\{A(k)\}_{k=1,2,\ldots}$ are independent and identically distributed (i.i.d) random matrices, we can define $\bar{A} \triangleq \mathbb{E}[A(k)]$ for all $k > 0$. With the edge set induced by the positive elements of $\bar{A}$, i.e.,

$$\bar{\mathcal{E}} = \{(j,i)|\bar{A}_{ij} > 0\}, \tag{2}$$

we define $\bar{G} = (\mathcal{V}, \bar{\mathcal{E}})$ as the mean connectivity graph, where $\mathcal{V}$ is the vertex set of the original network.

Now we give the following the assumption about $A(k)$:

Assumption 4. *(a) $\{A(k)\}_{k=1,2,\ldots}$ are i.i.d.;*
(b) There exists some constant $\gamma \in (0,1)$ such that $A(k)_{ii} \geq \gamma$ with probability 1 for all i and $\min_{j,i\in\bar{\mathcal{E}}} \frac{\bar{A}_{ij}}{2} \geq \gamma$;
(c) The mean connectivity graph is strongly-connected;
(d) $A(k)$ is doubly stochastic with probability 1 for all $k > 0$, i.e., $\sum_i a_{ij}^k = \sum_j a_{ij}^k = 1$ for any i and j.

Remark 1. *A simple weight matrix satisfying Assumption 4 is called the lazy Metropolis update [27], which is defined as follows:*

$$a_{ij}^k = \begin{cases} \frac{1}{2\max\{d_i,d_j\}} & \text{if } j \in N(i) \\ 1 - \sum_{j\in N(i)} \frac{1}{2\max\{d_i,d_j\}} & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

for all $k > 0$, where d_i is the degree of Processor i. In this case $A(k)$ is deterministic. The mean connectivity graph is strongly-connected if and only if the underlying graph of the network is strongly-connected. Using this matrix requires each processor to broadcast its degree to its neighbors along with its value.

Now we will give three theorems related to the convergence results of PUSSD. These theorems can still hold if the order of communication and computation operations in Algorithm 1 is exchanged because of their independence. Here we define two values used in the theorems: $B = (3+\frac{2}{\gamma^{2(n-1)}})\exp\{-\frac{\gamma^{4(n-1)}}{2}\}$ and $\theta = \exp\{-\frac{\gamma^{4(n-1)}}{4(n-1)}\}$, where γ is defined in Assumption 4.

Theorem 1. *If Assumption 1 and Assumption 4 are satisfied, then for Algorithm 1 with output Option II we have:*
(a) when $\eta_k = \eta = \frac{1}{\sqrt{K}}$ is fixed,

$$\mathbb{E}[f(v_i^K) - f(x^*)] \leq \frac{1}{\sqrt{K}}\left(\frac{||y^0 - x^*||_2^2}{2p} + \frac{2(1-p+np+\frac{1}{2}n)C^2}{n} + \frac{8npBC^2}{1-\theta}\right) \tag{4}$$

after K iterations for any i, where $y^0 = \frac{1}{n}\sum_{i=1}^n x_i^0$;
(b) when $\sum_{k=1}^\infty \eta_k = \infty$ and $\sum_{k=1}^\infty \eta_k^2 < \infty$,

$$\lim_{K\to\infty} \mathbb{E}[f(v_i^K) - f(x^*)] = 0 \tag{5}$$

for any i.

Proof. See Appendix A for details. □

Remark 2. *The last term of equation* (4) *are called consensus error because it is brought by the processor disagreement after K iterations. We can see that it is determined by the weight matrix and the topology of the network via B and θ.*

In Theorem 1, if we let the right side of (4) equal to ε and solve K from the equation, then we can get the the maximum iterations to achieve ε-suboptimality, which is written as (we allow K to be non-integer for simplicity):

$$K = \frac{1}{\varepsilon^2}\left(\frac{||y^0 - x^*||_2^2}{2p} + \frac{2(1-p+np+\frac{1}{2}n)C^2}{n} + \frac{8npBC^2}{1-\theta}\right)^2.$$

On average, np processors are computing the gradients in each iteration. Define the network computation cost of distributed optimization as the sum of the computation cost brought by the processors which compute the subgradients in the network. Then the expected network computation cost to achieve ε-suboptimality is bounded by

$$\text{Ecomp}(\varepsilon) \leq \frac{np}{\varepsilon^2}\left(\frac{1}{2p}||y^0 - x^*||_2^2 + \frac{2(1-p+np+\frac{1}{2}n)C^2}{n} + \frac{8npBC^2}{1-\theta}\right)^2. \tag{6}$$

where $p \in (0,1]$. Since the traditional DSSA corresponds to our algorithm with $p = 1$, then its expected network computation cost is bounded by

$$\text{Ecomp}(\varepsilon) \leq \frac{n}{\varepsilon^2}\left(\frac{1}{2}||y^0 - x^*||_2^2 + 3C^2 + \frac{8nBC^2}{1-\theta}\right)^2, \tag{7}$$

which can be also obtained following the proof line in [5]. Now we fix B and θ by using the same weight matrix for different p. Then from (6) and (7) we can see that $p = 1$ is not necessarily the best choice in terms of computation cost. If we know cost functions of other concurrent tasks in terms of p, we can minimize the total cost function by choosing an appropriate p.

Now we turn to the result for strongly convex functions.

Theorem 2. *If Assumption 1 ,3 and 4 are satisfied, then for Algorithm 1 with output Option III and the diminishing stepsize $\eta_k = \frac{2}{\mu p(k+1)}$,*

$$\mathbb{E}[f(z_i^K) - f(x^*)] \leq \frac{1}{K+1}\left(\frac{8(1-p+np+\frac{1}{2}n)C^2}{n\mu p} + \frac{16nBC^2}{(1-\theta)\mu} + \frac{8npBC^2}{K(1-\theta)}\right) \tag{8}$$

$\overset{K\to\infty}{\longrightarrow} 0.$

Proof. See Appendix B for details. □

From Theorem 2, we can see that our algorithm has a convergence rate of $O(1/K)$, which matches the order of the

convergence rate of centralized stochastic gradient descent [2]. When K is large, the last term of (8) can be neglected. Then using the same procedure of deriving (6), we can write the bound of the expected network computation cost to reach ε-suboptimality as

$$\begin{aligned}&\mathrm{Ecomp}(\varepsilon)\\ &\leq \frac{np}{\varepsilon}\left(\frac{8(1-p+np+\frac{1}{2}n)C^2}{n\mu p}+\frac{16nBC^2}{(1-\theta)\mu}\right)-np \end{aligned} \tag{9}$$

in this case. Again, $p=1$ does not necessarily minimize the above bound. The expected network computation cost of distributed optimization using PUSSD can be lower than the traditional DSSA.

Last, we consider smooth and strongly convex functions.

Theorem 3. *If Assumption 1, 2, 3 and 4 are satisfied, then for Algorithm 1 with output Option I and the fixed stepsize* $\eta_k=\eta\in(\frac{L-\mu}{p\mu L},\frac{1}{p\mu})$,

$$\begin{aligned}\mathbb{E}[f(x_i^{K+1})-f(x^*)] &\leq \left(\frac{L}{\mu}-pL\eta\right)^K\left(f(y^0)-f(x^*)-R\right)\\ &\quad +R+\frac{np\eta BC^2}{1-\theta}+npC^2B\theta^{K-1}\end{aligned} \tag{10}$$

$$\overset{K\to\infty}{\longrightarrow} R+\frac{np\eta BC^2}{1-\theta} \tag{11}$$

for any i, where $y^0=\frac{1}{n}\sum_{i=1}^n x_i^0$, *and*

$$R=\frac{\frac{2\eta^2L(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}+\frac{3np^2LBC^2(\eta^2+\eta)}{1-\theta}}{1-\frac{L}{\mu}+pL\eta}.$$

Proof. See Appendix C for details. □

From Theorem 3, we can see that a part of the optimality gap decreases in a linear rate. When this part dominates the whole optimality gap, the algorithm "seems" to have a linear convergence rate, which can be observed in Section V-A.

From the above analysis, p should be chosen to minimize (6) or (9) for computation cost reduction. In practice, we also need to consider priorities (larger p for tasks of higher priority) and fairness (not making p too extreme) requirements of concurrent tasks. It is interesting to explore how to choose p adaptively in future work.

IV. PUSSD with Less Communication Overhead

In general, a fixed weight matrix like (3) is often used in distributed optimization methods, where all the processors are involved in the communication operation. However, this may not be efficient when the communication cost is very large. This situation may arise in the example of wireless sensor networks when some channels are in deep fading [28]. Assumption 4 allows us to choose some suitable weight matrix to reduce the communication cost by not utilizing all the links, but we need additional communication protocols to realize such a weight matrix, which increases communication overhead. In our distributed optimization framework, we can reduce such overhead by directly taking advantage of the randomness brought by p in PUSSD while realizing reduction of the communication cost. The modified algorithm, called PUSSD with less communication overhead, is shown in Algorithm 2.

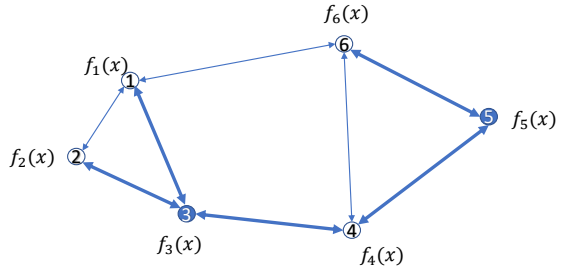


Fig. 1. The communication operation of Algorithm 2 in a network consisting of 6 processors in one iteration. The colored processors are active ones in this iteration and the bold lines are the links involved in the communication operation.

Here we call the processors which compute their subgradients the *active* processors and the others the *inactive* processors. Note that the inactive processors are actually processing other tasks and their inactivity is only with respect to the distributed optimization being performed. Define $N'(i)$ as the set of active neighbors for the inactive processors. In this algorithm, the communication only occurs between the active processors and their neighbors. For example, suppose Algorithm 2 is running in a network shown in Figure 1. If Processor 3 and Processor 5 choose to run the subgradient descent in this iteration, then the communication will happen in the links represented by bold lines in Figure 1. The inactive processors can recognize its active neighbors by the receipt of messages with source indicators (like TCP/IP [29]). A time window for communication operation can be set for the processors to guarantee the receipt of all messages that they should receive.

In the following theorem, we will show that Algorithm 2 has the same convergence results with Algorithm 1 based on the same assumptions. This theorem, however, can only hold with the order of computation and communication operations shown in Algorithm 2, because they are coupled by taking advantage of the same p, which prevents the direct application of proofs for Algorithm 1.

Theorem 4. *For Algorithm 2, Theorem 1, 2 and 3 still hold with their respective assumptions.*

Proof. One step of Algorithm 2 can be written as:

$$u_i^{k+1}=\begin{cases}\pi_\chi\{x_i^k-\eta_k\tilde{g}_i^k\} & \text{if } i\in I_{k+1}\\ x_i^k & \text{otherwise}\end{cases}$$

$$x_i^{k+1}=a_{ii}^k u_i^{k+1}+\sum_{j\in N'(i)}a_{ij}^k u_j^{k+1}.$$

Similar to (16), we can have

$$y^{k+1}=y^k+\frac{1}{n}\sum_{i\in I_{k+1}}(\Delta_i^k-\eta_k\tilde{g}_i^k).$$

with I_k replaced by I_{k+1}. Now, for step (23), we will take expectations to I_{k+1} conditioned on $\{x_i^k\}_{i=1}^n$ and then follow the same proof line to get the same convergence results for Algorithm 2. If the order of communication and computation operations in Algorithm 2 are exchanged, we need to take expectations to I_k in (23). Since I_k is coupled with $\{x_i^k\}_{i=1}^n$, we cannot obtain the bounds in (23) to continue the proof. □

As less processors are involved in the communication operation, the consensus process across the networks in Algorithm 2 may be slower than Algorithm 1. For example, in Algorithm 2 we can use a similar weight matrix to (3):

$$a_{ij}^k = \begin{cases} \frac{1}{2\max\{d_i^k, d_j^k\}} & \text{if } j \in N^k(i) \\ 1 - \sum_{j \in N^k(i)} \frac{1}{2\max\{d_i^k, d_j^k\}} & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where $N^k(i)$, d_i^k, d_j^k are defined with regard to $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$. Here $\mathcal{V}^k$ is the vertex set of the active processors and their neighbors in iteration k, and $\mathcal{E}^k$ is the edge set of the edges between the active processors and their neighbors. Using this matrix, the convergence rate of Algorithm 2 may be slower than Algorithm 1 because γ in Assumption 4 may decrease, leading to the increase of B and θ in the convergence results. Therefore for specific graphs and weight matrices, there should be a communication-computation cost tradeoff in Algorithm 2. In the Section V-A, we will use two network topologies to show that this tradeoff is related to the connectivity of networks.

As a common example of distributed optimization [27], we use Erdős-Renyi graph [30] to motivate the discussion of communication cost reduction in Algorithm 2 and examine the situations where this method is preferred. To that end, consider the $G(n, q)$ model where n processors connect with each other with probability q. We can choose $q = (1+\epsilon)\log(n)/n$ for some $\epsilon > 0$ to guarantee that the graph is strongly connected with high probability [30]. For this model we have the following proposition:

Proposition 1. *For Erdős-Renyi networks $G(n, q)$, the expected communication cost in each iteration of Algorithm 2 is $\frac{q}{2}(2p - p^2)(n^2 - n)\tau$.*

Proof. In Algorithm 2, a link is in communication for distributed optimization if and only if it exists and at least one of its two vertices is active. Then

$$\mathbb{P}\{\text{a link in communication}\} = q(1 - (1-p)^2) = q(2p - p^2).$$

So the number of the links in communication follows a binomial distribution $B\left(\binom{n}{2}, 2qp - qp^2\right)$. Now we can obtain the expected communication cost in one iteration:

$$\text{Ecomm} = \frac{q}{2}(2p - p^2)(n^2 - n)\tau.$$

□

In contrast, when all the processors are involved in the communication operation, the expected communication cost will be $\binom{n}{2}q\tau$. So the decrease is $\frac{q}{2}(p-1)^2(n^2-n)\tau$. If n and τ are large, this amount can be significant.

Algorithm 2 Partially Updated Subgradient Descent (PUSSD) with Less Communication Overhead for Processor i

Input: number of iterations K, probability to run subgradient descent p, stepsizes for K iterations $\{\eta_k\}_{k=1}^K$
Output: Option I: x_i^K; Option II: $v_i^K := \frac{\sum_{k=1}^K \eta_k x_i^k}{\sum_{k=1}^K \eta_k}$; Option III: $z_i^K := \frac{2}{K(K+1)} \sum_{k=1}^K k x_i^k$
Initialize $x_i^0 \in \chi$.
for $k = 1$ **to** K **do**
  Generate a random number r in $[0, 1]$.
  if $r < p$ **then**
    Compute a stochastic subgradient $\tilde{g}_i^{k-1}$ of $f_i(x_i^{k-1})$.
    Set $u_i^k \leftarrow \pi_\chi\{x_i^{k-1} - \eta_k \tilde{g}_i^{k-1}\}$.
    Send u_i^k to the neighbor processors.
    if received $\{u_j^k\}_{j \in N(i)}$ from all the neighbors. **then**
      Set $x_i^k \leftarrow a_{ii}^k u_i^k + \sum_{j \in N(i)} a_{ij}^k u_j^k$, where a_{ii}^k, a_{ij}^k satisfy Assumption 4.
    end if
  else
    Set $u_i^k \leftarrow x_i^{k-1}$ and process other tasks.
    if received $\{u_j^k\}_{j \in N'(i)}$ from all the active neighbor(s). **then**
      Send u_i^k to the active neighbor(s) and set $x_i^k \leftarrow a_{ii}^k u_i^k + \sum_{j \in N'(i)} a_{ij}^k u_j^k$ for some $a_{ii}^k \in [0, 1], a_{ij}^k \in [0, 1]$, where a_{ii}^k, a_{ij}^k satisfy Assumption 4.
    end if
  end if
end for

V. Numerical Results

In this section, we will show the advantages of PUSSD and its variant for three cases, l_2-norm loss, l_1-norm loss and l_2-norm loss with l_1 regularizer , in the regression setting. These objectives represent three types of functions we discussed in Section III: one satisfying Assumption 1, 2 and 3, one satisfying Assumption 1, and one satisfying Assumption 1 and 3. For the first case, we will conduct more comprehensive experiments to explore the performance of PUSSD and its variant, while the experiments of the last two cases are relatively brief.

A. l_2-norm Loss

Suppose each processor has a training set consisting of 20 data points $(x_{i,m}, y_{i,m})$, where $x_{i,m} \in \mathbb{R}^d$ and $y_{i,m} \in \mathbb{R}$ belong to the mth data point stored in Processor i. Then the empirical risk minimization for the l_2-norm loss is to solve the following optimization problem:

$$\min_{\theta \in \chi} f(\theta) = \frac{1}{20n} ||y - X^T\theta||_2^2, \tag{13}$$

where $X \in \mathbb{R}^{d \times 20n}$ is the matrix whose $(20(i-1)+m)$th column is $x_{i,m}$ and $y \in \mathbb{R}^{20n}$ is the vector whose $(20(i-1)+m)$th entry is $y_{i,m}$. $n(=100)$ is the number of processors in the

network. A stochastic gradient at θ of f_i is $x_{i,m}(x_{i,m}^T\theta - y_{i,m})$ by randomly choosing one data point $(x_{i,m}, y_{i,m})$ from the dataset of Processor i. It is easy to check that (13) satisfies Assumption 1, 2 and 3 when χ is compact.

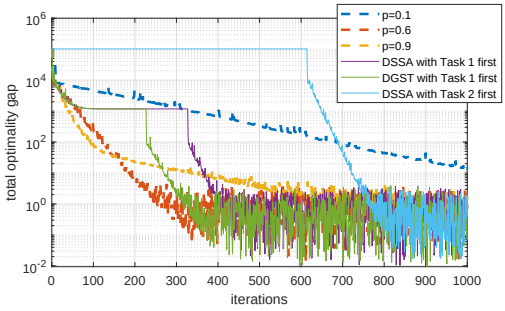
First, we consider that two tasks are processed by a network, which both solve Problem (13) with different datasets. The network topology is a connected 10-regular graph, where each node is connected with 10 nodes. For Task 1, we set $y_{i,m} = x_{i,m}^T\theta^* + \xi_{i,m}$, where θ^* is a 10-dimensional vector whose first 3 entries are $30, 20, 10$ and the remaining 0s. In this experiment, we generate $x_{i,m}$ by sampling from a uniform distribution in $[0, 10]^3$, and $\xi_{i,m} \in \mathbb{R}$ follows a uniform distribution in $[0, 5]$. For Task 2, we set θ^* to be a 10-dimensional vector whose first 3 entries are $3, 2, 1$ and the remaining 0s. The generations of $x_{i,m}$ and $y_{i,m}$ are the same with Task 1. In both tasks, we set χ to be $[-100, 100]^{10}$

When applying PUSSD (Algorithm 1) to these two tasks, we let each processor process Task 1 with probability p and Task 2 otherwise, both using (3) for the weight matrix. Meanwhile we use three sequential processing algorithms as comparisons: DSSA [14] with Task 1 first, DSSA with Task 2 first and DGST [20] with Task 1 first. For these schemes, when the optimality gap of one task is less than a certain value (which is set to be more than what PUSSD can reach within our iteration range), the network proceeds to process the other task. [1] The stepsizes of each method are chosen to achieve their satisfying performance. The optimality gap in iteration k is measured by $f(\theta_i^k) - f(\theta^*)$ (Option I in Algorithm 1), where θ_i^k is the output of Processor i in iteration k. $f(\theta^*)$ is calculated by an inbuilt Matlab function, which is also true for other experiments. Processor i is randomly chosen at the beginning of the experiment, and kept tracked afterwards.

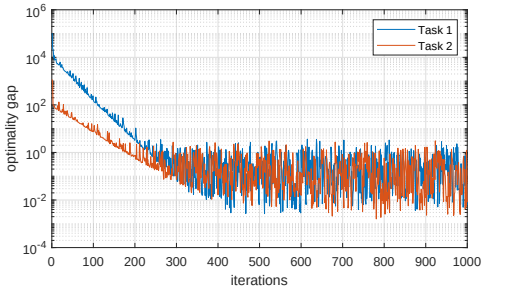
In Figure 2(a) we compare PUSSD using Theorem 3 with different p values with three sequential processing methods mentioned above. We set the initial point to be zero vector for each method. The metric is the total optimality gap of two tasks. From the figure we can see that larger p gives a faster rate in the initial phase. This is because Task 1 has a greater decrease in its optimality gap than Task 2 if processed alone, given that the initial point of Task 1 is much further away from the optimum. So more emphasis on Task 1 can benefit the convergence rate initially. Meanwhile, we can observe that when any of the tasks is close to its optimum, its convergence rate will decrease dramatically. Therefore putting more weight on any task is not a wise decision in this phase. As a result, PUSSD with extreme p values and the sequential processing algorithms are all impacted by the slow final phase. Given this tradeoff, $p = 0.6$ outperforms others after 150 iterations. For DSGT algorithm, it has a better performance than DSSA, but still worse than PUSSD with $p = 0.6$.

In Figure 2(b), we plot the evolution of two tasks when PUSSD with $p = 0.6$ is applied. From the figure, we can see that the optimality gaps of both tasks decrease together, so

[1] In practice each processor has no access to the optimality gap and terminates when the value changes little. The switching criteria used here is for simplicity and comparison.



(a) The comparison of PUSSD with $p = 0.1$, $p = 0.6$, $p = 0.9$ and the three sequential processing algorithms for l_2 norm loss



(b) The evolution of two tasks using PUSSD with $p = 0.6$ for l_2 norm loss

Fig. 2. PUSSD applied to two concurrent optimization tasks for l_2 norm loss

the fairness is also guaranteed. The decrease is approximately in a linear rate before reaching the final optimality gap, as explained after the statement of Theorem 3. This experiment shows the advantages of PUSSD when dealing with concurrent optimization problems for strongly convex, smooth functions.

To further explore the impact of p values on the performance, we run PUSSD with different p values for different iterations to see the total optimality gap that they can achieve. We plot the total optimality gap versus p values in Figure 3 after $K(= 100, 200, 300)$ iterations of PUSSD. We can see that for small iterations, larger p values like $0.8, 0.9$ can achieve smaller optimality gap, which demonstrates our observation of the initial phase in Figure 2(a). For more iterations of PUSSD, the preferred p values are towards to the middle of the range (like $0.6, 0.7$) because we need to balance the progress of two tasks without letting the final phase of any task drag the other task. This phenomenon motivates us to consider PUSSD with dynamic p values, which is left for future work.

Now we assume that Task 1 is processed along with other kinds of tasks and focus on the performance of Task 1. The measurement of the optimality gap is the same with the above experiment, but now only for Task 1. We compare Algorithm 2 using the weight matrix defined in (12) with Algorithm 1 using (3) for $p = 0.2$, $p = 0.4$ and $p = 0.6$ in Figure 4(a) where the network topology is a 10-regular graph. We can find that the convergence rates of two algorithms are very close to each other for all the choices of p. In this case, Algorithm 2 is better if we take communication cost into account.

Meanwhile, since the communication cost of Algorithm 2 is

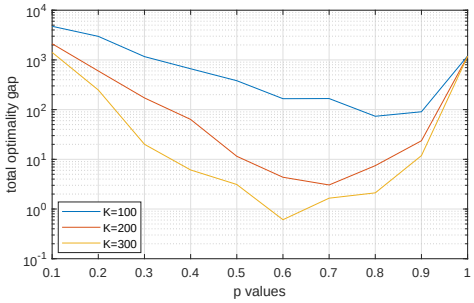


Fig. 3. The comparison of PUSSD with different p values after K iterations

sensitive to the network topology, we also test our algorithms in a star graph, where one processor is connected with the other 99 processors. It has much less connectivity than the 10-regular network which we considered above. It means for the same p, the number of active communication links of Algorithm 2 in a star network is much less than the one in a 10-regular network.

The result is shown in Figure 4(b). We can see the gaps between Algorithm 1 and Algorithm 2 are larger compared with Figure 4(a) because of less connectivity. Meanwhile, for a large p value, the gap becomes smaller. To further compare Algorithm 1 and 2 in the star network, we also plot the total cost of algorithms mentioned above versus communication cost per link τ in Figure 4(c). Here the total cost of an algorithm is its sum of computation cost and communication cost across the network to reach 0.5 optimality gap of Task 1. Recall that we set the computation cost of a stochastic gradient to be 1. We can see that for all the p values, when the communication cost is large enough, the total cost of Algorithm 2 is lower than Algorithm 1. It demonstrates the communication efficiency of Algorithm 2 compared with Algorithm 1.

B. l_1-norm Loss

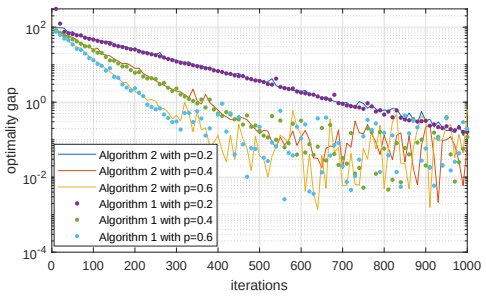
In this subsection, we consider an unconstrained empirical risk minimization for the l_1-norm loss with the same dataset in Section V-A:

$$\min_{\theta\in\mathbb{R}^d} f(\theta) = \frac{1}{20n}||y - X^T\theta||_1. \tag{14}$$

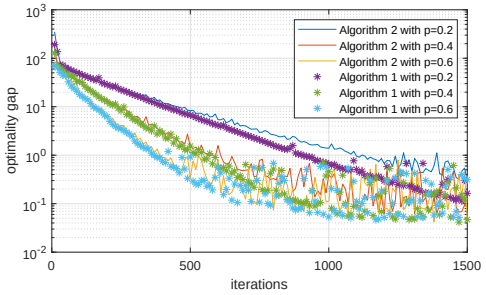
It is easy to check the above function satisfies Assumption 1.

Similarly, we first consider the case where two tasks are processed by a 10-regular network. We refer Task 3 and 4 to be the tasks solving (14) with the same dataset of Task 1 and Task 2, respectively. The performance metric is similar to Section V-A.

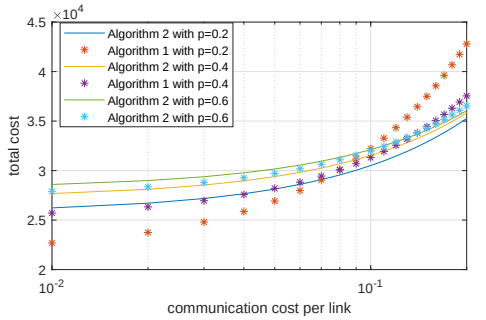
In Figure 5(a) we compare PUSSD using Theorem 1(a) with different p values with three sequential processing algorithms: DSSA with Task 3 first, DSSA with Task 4 first and DDA [6] with Task 4 first. The switching criteria of the sequential algorithms are similar to Section V-A. From the figure we can see that PUSSD with $p = 0.6$ and $p = 0.9$ give the fastest rate at the initial phase. After that $p = 0.6$ and DDA with Task 4



(a) The comparison of Algorithm 1 and Algorithm 2 for a 10-regular network



(b) The comparison of Algorithm 1 and Algorithm 2 for a star network



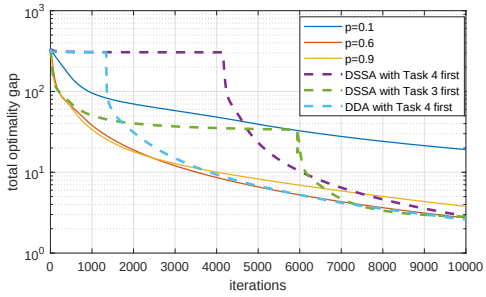
(c) The total cost of Algorithm 1 and 2 to reach 0.5 optimality gap with different communication cost per link in a star network

Fig. 4. The comparison of Algorithm 1 and 2 for different network topologies

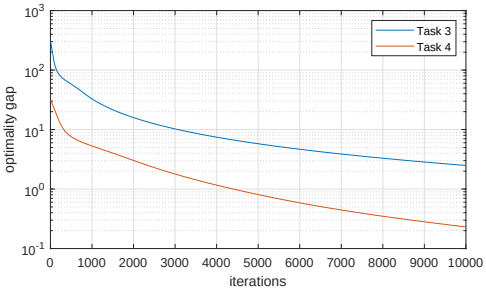
first have similar rates, which is faster than others. Accounting for the whole process, PUSSD with $p = 0.6$ is the best choice. From Figure 5(b) we can also see that the optimality gaps of both tasks decrease concurrently when we apply PUSSD with $p = 0.6$, so the fairness is maintained with this choice. This experiment shows the advantages of PUSSD when dealing with concurrent optimization problems for nonsmooth functions.

Now we assume that Task 4 is processed along with other kinds of tasks in a 10-regular network and study the performance of Task 4. We apply Algorithm 2 to Task 4 with the weight matrix defined in (12). We compare this algorithm with Algorithm 1 using (3) for different p to show how the computation cost is impacted.

In Figure 6, we plot the optimality gap of Task 4 versus iterations for Algorithm 1 and Algorithm 2 when $p = 0.2$, $p = 0.6$ and $p = 1$ (which is also the traditional DSSA). We can see that for the same p, the performance of the two algorithms is almost the same. So similar to Figure 4(a), the decrease



(a) The comparison of PUSSD with $p = 0.1$, $p = 0.6$, $p = 0.9$ and the three sequential processing algorithms for l_1 norm loss



(b) The evolution of two tasks using PUSSD with $p = 0.6$ for l_1 norm loss

Fig. 5. PUSSD applied to two concurrent optimization tasks for l_1 norm loss

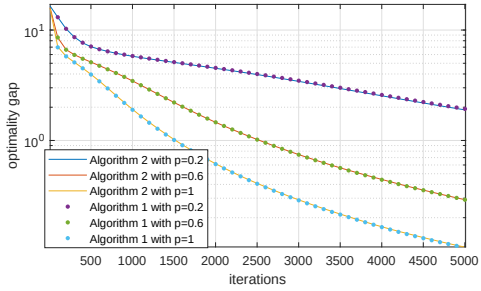


Fig. 6. The comparison of Algorithm 1 and Algorithm 2 with different p for l_1 norm loss

of communication does not impact the convergence rate too much when Algorithm 2 is applied. In this case, Algorithm 2 can reduce the communication cost while not increasing the computation cost of Task 4, which is preferred when the system has high communication costs. Meanwhile we can compare the computation cost of PUSSD with the traditional DSSA ($p = 1$) to reach a certain gap in this figure. For example, when the optimality gap is 10^0, PUSSD with $p = 0.6$ needs about 2500 iterations, while the traditional DSSA needs about 1500 iterations. So the expected network computation cost to reach 10^0 gap is almost the same for PUSSD and the traditional DSSA (since $2500 \times 0.6 = 1500$). But our algorithm allows for the parallel processing of other tasks.

C. l_2-norm Loss with l_1 Regularizer

For the completeness of our simulation, we also consider the example of LASSO regression [31], where we add the l_1 regularizer to (14) for the sparsity of θ. In this simulation, the objective function is as follows:

$$\min_{\theta \in \chi} f(\theta) = \frac{1}{20n}||y - X^T\theta||_2^2 + 0.1||\theta||_1, \qquad (15)$$

where $\chi = [-100, 100]^{10}$. When we use the datasets mentioned in Section V-A, it can be proved that (15) satisfies Assumption 1 and 3.

For brevity, we only consider the case where two tasks minimizing the above objective function are processed by a 10-regular network in this subsection. They are referred as Task 5 and 6 with the datasets same with Task 1 and 2, respectively. We compare PUSSD using Theorem 2 and different p values with two sequential processing algorithms: the method in [18] with Task 5 first and with Task 6 first. The result is shown in Figure 7(a). We can see that PUSSD with $p = 0.6$ is still the best among these methods. Here except the reason mentioned in the previous two subsections, we can observe another advantage of PUSSD. Because both PUSSD using Theorem 2 and the method in [18] adopt diminishing stepsizes, the stepsize of the first iteration can be too large, which causes a jump in function values. Obviously, this kind of jump is not good for convergence. For the sequential processing algorithms, there is a jump at the start of each task, while PUSSD only has one jump at the beginning of the whole algorithm. So the jump has less harm for the convergence of PUSSD. Meanwhile in Figure 7(b), we can see that the optimality gaps of two tasks decrease concurrently for PUSSD with $p = 0.6$, which maintains the fairness of two tasks. Through this experiment, we also demonstrate the advantages of PUSSD for strongly convex functions.

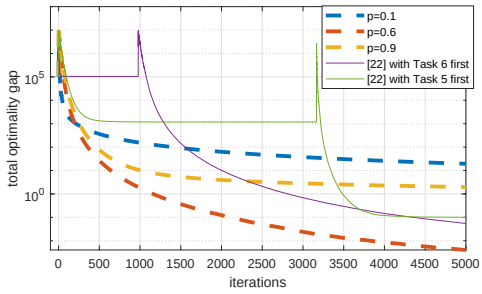
VI. Conclusion

In this paper we proposed Partially Updated Subgradient Descent (PUSSD) to enable the split of computing resources for distributed optimization and concurrent tasks in networks. We then derived the convergence results of PUSSD with different assumptions on each function, which showed its favorable convergence and convergence rate characteristics. For one possible application situation, we developed PUSSD with less communication overhead to get a better performance in communication costs. The experiments of three machine learning problems demonstrated the flexibility and efficiency of our algorithms along with their ability to share computing resources more fairly amongst concurrent tasks.

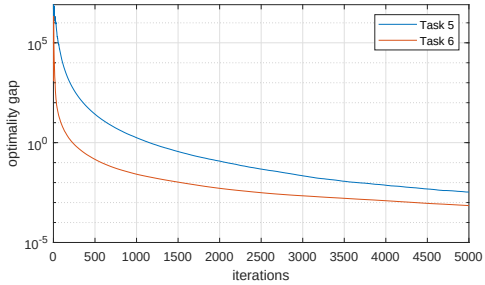
References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] L. Bottou, F. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.



(a) The comparison of PUSSD with $p = 0.1$, $p = 0.6$, $p = 0.9$ and two sequential processing algorithms for LASSO



(b) The evolution of two tasks using PUSSD with $p = 0.6$ for LASSO

Fig. 7. PUSSD applied to two concurrent optimization tasks for LASSO

[3] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

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

[5] I. Lobel and A. Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1291–1306, 2011.

[6] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic control*, vol. 57, no. 3, pp. 592–606, 2012.

[7] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012, pp. 5453–5458.

[8] 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 Processing*, vol. 62, no. 7, pp. 1750–1761, 2014.

[9] A. Makhdoumi and A. Ozdaglar, "Convergence rate of distributed admm over networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 5082–5095, 2017.

[10] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, "Optimal algorithms for smooth and strongly convex distributed optimization in networks," *arXiv preprint arXiv:1702.08704*, 2017.

[11] G. Qu and N. Li, "Accelerated distributed nesterov gradient descent," *arXiv preprint arXiv:1705.07176*, 2017.

[12] R. Tutunov, H. B. Ammar, and A. Jadbabaie, "A distributed newton method for large scale consensus optimization," *arXiv preprint arXiv:1606.06593*, 2016.

[13] A. Mokhtari, Q. Ling, and A. Ribeiro, "Network newton distributed optimization methods," *IEEE Transactions on Signal Processing*, vol. 65, no. 1, pp. 146–161, 2017.

[14] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *Journal of optimization theory and applications*, vol. 147, no. 3, pp. 516–545, 2010.

[15] A. Nedić and A. Olshevsky, "Stochastic gradient-push for strongly convex functions on time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 3936–3947, 2016.

[16] G. Lan, S. Lee, and Y. Zhou, "Communication-efficient algorithms for decentralized and stochastic optimization," *arXiv preprint arXiv:1701.03961*, 2017.

[17] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[18] M. O. Sayin, N. D. Vanli, S. S. Kozat, and T. Basar, "Stochastic subgradient algorithms for strongly convex optimization over distributed networks," *IEEE Transactions on Network Science and Engineering*, vol. 4, no. 4, pp. 248–260, Oct 2017.

[19] D. Jakovetic, D. Bajovic, A. K. Sahu, and S. Kar, "Convergence rates for distributed stochastic optimization over random networks," *arXiv preprint arXiv:1803.07836*, 2018.

[20] S. Pu and A. Nedić, "Distributed stochastic gradient tracking methods," *Mathematical Programming*, pp. 1–49, 2020.

[21] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, 2018, pp. 5973–5983.

[22] S. Shi, K. Zhao, Q. Wang, Z. Tang, and X. Chu, "A convergence analysis of distributed sgd with communication-efficient gradient sparsification." in *IJCAI*, 2019, pp. 3411–3417.

[23] S. Shi, Z. Tang, Q. Wang, K. Zhao, and X. Chu, "Layer-wise adaptive gradient sparsification for distributed deep learning with convergence guarantees," *arXiv preprint arXiv:1911.08727*, 2019.

[24] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, 2018, pp. 4447–4458.

[25] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 1299–1309.

[26] S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, and X. Chu, "A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 2238–2247.

[27] A. Nedić, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *arXiv preprint arXiv:1709.08765*, 2017.

[28] T. Watteyne, S. Lanzisera, A. Mehta, and K. S. Pister, "Mitigating multipath fading through channel hopping in wireless sensor networks," in *2010 IEEE International Conference on Communications*. IEEE, 2010, pp. 1–5.

[29] W. Stallings, *High-speed networks: TCP/IP and ATM design principles*. Prentice hall Englewood Cliffs, NJ, 1998, vol. 172.

[30] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[31] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[32] D. P. Bertsekas, *Nonlinear programming*. Athena scientific Belmont, 2016.

Throughout the appendix, $||\,.\,||$ refers to l_2 norm, and I_k is the set of processors computing (sub)gradients in iteration k.

APPENDIX A
PROOF OF THEOREM 1

Suppose $k \geq 1$. The output of Processor i in iteration $k+1$ in Algorithm 1 is

$$x_i^{k+1} = a_{ii}^{k+1} u_i^k + \sum_{j \in N(i)} a_{ij}^{k+1} u_j^k,$$

where $u_i^k = \begin{cases} \pi_\chi\{x_i^k - \eta_k \tilde{g}_i^k\} & \text{if } i \in I_k \\ x_i^k & \text{otherwise} \end{cases}$ and $\tilde{g}_i^k$ is a stochastic subgradient of $f_i(x_i^k)$ and $\mathbb{E}[\tilde{g}_i^k] = g_i^k \in \partial f_i(x_i^k)$. Now we define $y^{k+1} = \frac{1}{n}\sum_{i=1}^n x_i^{k+1}, v_i^k = x_i^k - \eta_k \tilde{g}_i^k, \Delta_i^k = \pi_\chi\{v_i^k\} - v_i^k$.

Since $A(k+1)$ is doubly stochastic with probability 1, we have

$$\begin{aligned} y^{k+1} &= \frac{1}{n}\sum_{i=1}^{n}[a_{ii}^{k+1}u_i^k + \sum_{j\in N(i)} a_{ij}^{k+1}u_j^k] \\ &= \frac{1}{n}\sum_{i=1}^{n} u_i^k \\ &= \frac{1}{n}\sum_{i=1}^{n} x_i^k + \frac{1}{n}\sum_{i\in I_k}(\Delta_i^k - \eta_k\tilde{g}_i^k) \\ &= y^k + \frac{1}{n}\sum_{i\in I_k}(\Delta_i^k - \eta_k\tilde{g}_i^k). \end{aligned} \tag{16}$$

Define x^* as any point in the optimality set S of the original problem. Then

$$\begin{aligned} ||y^{k+1} - x^*||^2 = ||y^k - x^*||^2 + \frac{1}{n^2}||\sum_{i\in I_k}(\Delta_i^k - \eta_k\tilde{g}_i^k)||^2 \\ + \frac{2}{n}\sum_{i\in I_k}(\Delta_i^k - \eta_k\tilde{g}_i^k)^T(y^k - x^*). \end{aligned} \tag{17}$$

Now we look at the second term in equation (17).

$$\frac{1}{n^2}||\sum_{i\in I_k}(\Delta_i^k - \eta_k\tilde{g}_i^k)||^2 \le \frac{1}{n^2}(\sum_{i\in I_k}||\Delta_i^k|| + \sum_{i\in I_k}\eta_k||\tilde{g}_i^k||)^2.$$

Since χ is a convex set and $u_i^k \in \chi$, then $x_i^{k+1} \in \chi$ because it is a convex combination of u_i^k. Similarly, we have $x_i^k \in \chi$. From the definition of projection operation, $\pi_{\mathcal{X}}\{v_i^k\}$ has the smallest distance to v_i^k among the points belonging to χ. Since $x_i^k \in \chi$, we have

$$\begin{aligned} ||\Delta_i^k|| &= ||\pi_{\mathcal{X}}\{v_i^k\} - v_i^k|| \\ &\le ||x_i^k - v_i^k|| \\ &= \eta_k||\tilde{g}_i^k|| \le \eta_k C. \end{aligned} \tag{18}$$

Meanwhile, from (18) and Assumption 1, we have

$$\begin{aligned} \frac{1}{n^2}||\sum_{i\in I_k}(\Delta_i^k - \eta_k\tilde{g}_i^k)||^2 &\le \frac{1}{n^2}(\sum_{i\in I_k}||\Delta_i^k|| + \sum_{i\in I_k}\eta_k||\tilde{g}_i^k||)^2 \\ &\le \frac{4}{n^2}(\sum_{i\in I_k}\eta_k||\tilde{g}_i^k||)^2 \\ &\le \frac{4\eta_k^2|I_k|^2C^2}{n^2}. \end{aligned} \tag{19}$$

where $|I_k|$ is the cardinality of the set I_k. Now we look at the last term in equation (17). We notice that

$$\begin{aligned} (\Delta_i^k)^T(y^k - x^*) &= (\Delta_i^k)^T(y^k - v_i^k) + (\Delta_i^k)^T(v_i^k - x^*) \\ &\le ||\Delta_i^k||(||y^k - x_i^k|| + \eta_k||\tilde{g}_i^k||) \\ &\quad + (\Delta_i^k)^T(v_i^k - x^* - \Delta_i^k + \Delta_i^k) \\ &\le \eta_k C||y^k - x_i^k|| + \eta_k^2C^2 \\ &\quad + (\Delta_i^k)^T(v_i^k - x^* - \Delta_i^k + \Delta_i^k) \qquad (20) \\ &= \eta_k C||y^k - x_i^k|| + \eta_k^2C^2 - ||\Delta_i^k||^2 \\ &\quad + (v_i^k - \pi_\chi\{v_i^k\})^T(x^* - \pi_\chi\{v_i^k\}) \\ &\le \eta_k C||y^k - x_i^k|| + \eta_k^2C^2, \qquad (21) \end{aligned}$$

where the inequality (20) is from (18) and the inequality (21) is from $(v_i^k - \pi_\chi\{v_i^k\})^T(x^* - \pi_\chi\{v_i^k\}) \le 0$ due to the projection theorem (Proposition 1.1.4 of [32]).

Define $D_k = \max_{i\in\{1,2,...,n\}}||y^k - x_i^k||$. Then we have

$$\frac{2}{n}\sum_{i\in I_k}(\Delta_i^k)^T(y^k - x^*) \le \frac{2|I_k|}{n}(\eta_k C D_k + \eta_k^2C^2). \tag{22}$$

Taking expectation to both sides of (17) conditioned on $\{x_i^k\}_{i=1}^n$ along with (19) and (22), we have

$$\begin{aligned} &\mathbb{E}[||y^{k+1} - x^*||^2|\{x_i^k\}_{i=1}^n] \\ &\le ||y^k - x^*||^2 + \frac{4\eta_k^2(np(1-p) + n^2p^2)C^2}{n^2} \\ &\quad + 2p(\eta_k C D_k + \eta_k^2C^2) - \frac{2p\eta_k}{n}\sum_{i=1}^{n}(g_i^k)^T(y^k - x^*), \end{aligned} \tag{23}$$

where $\mathbb{E}[|I_k||\{x_i^k\}_{i=1}^n] = np$ and $\mathbb{E}[|I_k|^2|\{x_i^k\}_{i=1}^n] = np(1-p) + n^2p^2$ by the properties of the binomial distribution. For the last term of equation (23), we have the following bound:

$$\begin{aligned} &-\frac{p\eta_k}{n}\sum_{i=1}^{n}(g_i^k)^T(y^k - x^*) \\ &= -\frac{p\eta_k}{n}\sum_{i=1}^{n}(g_i^k)^T(x_i^k - x^* + y^k - x_i^k) \\ &\le -\frac{p\eta_k}{n}\sum_{i=1}^{n}(f_i(x_i^k) - f_i(x^*)) + \frac{p\eta_k}{n}\sum_{i=1}^{n}||g_i^k||||y^k - x_i^k|| \qquad (24) \\ &\le -\frac{p\eta_k}{n}\sum_{i=1}^{n}(f_i(x_i^k) - f_i(x^*)) + p\eta_k C D_k, \qquad (25) \end{aligned}$$

where (24) is from the definition of subgradients and the Cauchy-Schwartz inequality, (25) is from $||g_i^k|| \le \mathbb{E}||\tilde{g}_i^k|| < C$ by Assumption 1 and Jensen's inequality.

For any $j \in \{1, ..., n\}$

$$\begin{aligned} &-\frac{p\eta_k}{n}\sum_{i=1}^{n}(f_i(x_i^k) - f_i(x^*)) \\ &= -\frac{p\eta_k}{n}\sum_{i=1}^{n}(f_i(x_j^k) - f_i(x^*) + f_i(x_i^k) - f_j(x_i^k)) \\ &\le -p\eta_k(f(x_j^k) - f(x^*)) + \frac{p\eta_k}{n}C\sum_{i=1}^{n}||x_i^k - x_j^k|| \\ &\le -p\eta_k(f(x_j^k) - f(x^*)) + 2p\eta_k C D_k, \qquad (26) \end{aligned}$$

where (26) is from $||x_i^k - x_j^k|| \le ||y^k - x_i^k|| + ||y^k - x_j^k|| \le 2D_k$. Combining all these results, we have

$$\begin{aligned} &\mathbb{E}[||y^{k+1} - x^*||^2|\{x_i^k\}_{i=1}^n] \\ &\le ||y^k - x^*||^2 + \frac{4\eta_k^2(p(1-p) + np^2 + \frac{1}{2}np)C^2}{n} \\ &+ 8p\eta_k C D_k - 2p\eta_k(f(x_j^k) - f(x^*)). \end{aligned} \tag{27}$$

Now taking expectations over $\{x_i^k\}_{i=1}^n$, we have:

$$\begin{aligned}&\mathbb{E}[||y^{k+1}-x^*||^2]\\&\leq \mathbb{E}[||y^k-x^*||^2]+\frac{4\eta_k^2(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}\\&+8p\eta_k C\mathbb{E}D_k-2p\eta_k\mathbb{E}[f(x_j^k)-f(x^*)].\end{aligned}\tag{28}$$

Now we want to bound $\mathbb{E}D_k$. First, for $k\geq 2$, one step of the algorithm can be written as

$$x_i^k=\sum_{j=1}^n[A(k)]_{ij}(x_j^{k-1}+\zeta_j^{k-1}),$$

where

$$\zeta_j^{k-1}=(\pi_\chi\{x_j^{k-1}-\eta_k\tilde{g}_j^{k-1}\}-x_j^{k-1})\mathbb{1}_{j\in I_{k-1}}$$

Define

$$\Phi(k,s)=A(k)A(k-1)...A(s)$$

for $k>s$ and $\Phi(k,k)=A(k)$. Then by induction we have

$$x_i^k=\sum_{j=1}^n[\Phi(k,1)]_{ij}x_j^0+\sum_{r=2}^k\sum_{j=1}^n[\Phi(k,r)]_{ij}\zeta_j^{r-1}.$$

Meanwhile, as $A(k)$ is doubly stochastic with probability 1 for any k and $\frac{1}{n}\sum_{i=1}^n x_i^1=\frac{1}{n}\sum_{i=1}^n x_i^0$, we have

$$\begin{aligned}y^k&=y^{k-1}+\frac{1}{n}\sum_{j=1}^n\zeta_j^{k-1}\\&=\frac{1}{n}\sum_{j=1}^n x_j^0+\frac{1}{n}\sum_{r=2}^k\sum_{j=1}^n\zeta_j^{r-1}.\end{aligned}$$

So we have

$$\begin{aligned}&\mathbb{E}||y^k-x_i^k||\\&=\mathbb{E}||\sum_{j=1}^n x_j^0(\frac{1}{n}-[\Phi(k,1)]_{ij})+\sum_{r=2}^k\sum_{j=1}^n(\frac{1}{n}-[\Phi(k,r)]_{ij})\zeta_j^{r-1}||\end{aligned}\tag{29}$$

$$\begin{aligned}&\leq\max_{1\leq j\leq n}||x_j^0||\sum_{j=1}^n\mathbb{E}|\frac{1}{n}-[\Phi(k,1)]_{ij}|\\&+\sum_{r=2}^k\sum_{j=1}^n\mathbb{E}[|\frac{1}{n}-[\Phi(k,r)]_{ij}|||\zeta_j^{r-1}||]\end{aligned}\tag{30}$$

$$\begin{aligned}&=\max_{1\leq j\leq n}||x_j^0||\sum_{j=1}^n\mathbb{E}|\frac{1}{n}-[\Phi(k,1)]_{ij}|\\&+\sum_{r=2}^k\sum_{j=1}^n\mathbb{E}|\frac{1}{n}-[\Phi(k,r)]_{ij}|\mathbb{E}||\zeta_j^{r-1}||,\end{aligned}\tag{31}$$

where (30) is from Cauchy-Schwartz inequality and (31) is because $\Phi(k,r)$ and ζ_j^{r-1} are independent. By the projection theorem (Proposition 1.1.4 of [32]), we have

$$\begin{aligned}\mathbb{E}||\zeta_j^k||&\leq\mathbb{E}[||(\pi_\chi\{x_j^k-\eta_k\tilde{g}_j^k\}-x_j^k)||||\mathbb{1}_{j\in I_k}||]\\&\leq\mathbb{E}[||\eta_k\tilde{g}_j^k||||\mathbb{1}_{j\in I_k}||]\\&\leq\eta_k Cp.\end{aligned}$$

Meanwhile, define $b(k,s)=\max_{(i,j)\in 1...n}|\frac{1}{n}-\Phi(k,s)_{ij}|$. From Lemma 7 in [5], when Assumption 4 is satisfied, we have

$$\mathbb{E}[b(k,s)]\leq B\theta^{k-s},\tag{32}$$

where $B=(3+\frac{2}{\gamma^{2(n-1)}})\exp(-\frac{\gamma^{4(n-1)}}{2})$, $\theta=\exp(-\frac{\gamma^{4(n-1)}}{4(n-1)})<1$.

So we have for $k\geq 2$ and any $i\in\{1,2,...,n\}$

$$\begin{aligned}\mathbb{E}||y^k-x_i^k||&\leq n(\max_{1\leq j\leq n}||x_j^0||)B\theta^{k-1}+\sum_{r=2}^k npB\theta^{k-r}\eta_{r-1}C\\&\leq np\sum_{r=1}^k\eta_{r-1}CB\theta^{k-r}\end{aligned}\tag{33}$$

by assuming $\max_{1\leq j\leq n}||x_j^0||\leq pC$ (we can always enlarge C to make this assumption satisfied) and $\eta_0=1$.

Thus $\mathbb{E}D_k\leq np\sum_{r=1}^k\eta_{r-1}CB\theta^{k-r}$. Plugging it into (28), we have

$$\begin{aligned}&\mathbb{E}[||y^{k+1}-x^*||^2]\\&\leq\mathbb{E}||y^k-x^*||^2+\frac{4\eta_k^2(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}\\&+8np^2\eta_k C^2\sum_{r=1}^k\eta_{r-1}B\theta^{k-r}-2p\eta_k\mathbb{E}[f(x_j^k)-f(x^*)].\end{aligned}$$

Telescoping from $k=1$ to K and doing simple manipulations, we have

$$\begin{aligned}&\mathbb{E}[\frac{\sum_{k=1}^K\eta_k(f(x_j^k)-f(x^*))}{\sum_{k=1}^K\eta_k}]\\&\leq\frac{||y^0-x^*||^2}{2p\sum_{k=1}^K\eta_k}+\frac{\sum_{k=1}^K\eta_k^2}{\sum_{k=1}^K\eta_k}\cdot\frac{2(1-p+np+\frac{1}{2}n)C^2}{n}\\&+\frac{4npBC^2\sum_{k=1}^K\eta_k\sum_{r=1}^k\eta_{r-1}\theta^{k-r}}{\sum_{k=1}^K\eta_k},\end{aligned}\tag{34}$$

where $y^1=y^0=\frac{1}{n}\sum_{i=1}^n x_i^0$.

(a) If $\eta_k=\eta=\frac{1}{\sqrt{K}}$ for some constant K, then from (34) we have

$$\begin{aligned}&\sum_{k=1}^K\frac{1}{K}\mathbb{E}[f(x_j^k)-f(x^*)]\\&\leq\frac{1}{\sqrt{K}}(\frac{||y^0-x^*||^2}{2p}+\frac{2(1-p+np+\frac{1}{2}n)C^2}{n}+\frac{8npBC^2}{1-\theta}).\end{aligned}$$

(b) If $\sum_{k=1}^\infty\eta_k=\infty$ and $\sum_{k=1}^\infty\eta_k^2<\infty$, then

$$\sum_{k=1}^K\eta_k\sum_{r=1}^k\eta_{r-1}\theta^{k-r}\leq\sum_{k=1}^K\sum_{r=1}^k\eta_{r-1}^2\theta^{k-r}.\tag{35}$$

By applying Lemma 9 in Lobel et al.'s work [5] to the right hand of (35), we have

$$\lim_{K\to\infty}4npBC^2\sum_{k=1}^K\eta_k\sum_{r=1}^k\eta_{r-1}\theta^{k-r}<\infty.$$

So from (34) we have

$$\lim_{K\to\infty}\mathbb{E}[\frac{\sum_{k=1}^K \eta_k(f(x_j^k)-f(x^*))}{\sum_{k=1}^K\eta_k}]=0.$$

Since f is convex, then $\frac{\sum_{k=1}^K \eta_k(f(x_j^k)-f(x^*))}{\sum_{k=1}^K\eta_k} \geq f(\frac{\sum_{k=1}^K\eta_k x_j^k}{\sum_{k=1}^K\eta_k})-f(x^*)$, which gives the final result.

Appendix B
Proof of Theorem 2

Our proof will start from (23). Since f_i is μ-strongly convex, we have the new bound for the last term:

$$\begin{aligned}
&-\frac{p\eta_k}{n}\sum_{i=1}^n(g_i^k)^T(y^k-x^*)\\
&=-\frac{p\eta_k}{n}\sum_{i=1}^n(g_i^k)^T(x_i^k-x^*+y^k-x_i^k)\\
&\leq-\frac{p\eta_k}{n}\sum_{i=1}^n(f_i(x_i^k)-f_i(x^*)+\frac{\mu}{2}||x_i^k-x^*||^2)\\
&\quad+\frac{p\eta_k}{n}\sum_{i=1}^n||g_i^k||||y^k-x_i^k|| \qquad (36)\\
&\leq-\frac{p\eta_k}{n}\sum_{i=1}^n(f_i(y^k)-f_i(x^*)+\frac{\mu}{2}||x_i^k-x^*||^2\\
&\quad+f_i(x_i^k)-f_i(y^k))+p\eta_kCD_k\\
&\leq-\frac{p\eta_k}{n}\sum_{i=1}^n(f_i(y^k)-f_i(x^*)+\frac{\mu}{2}||x_i^k-x^*||^2\\
&\quad+(\bar{g}^k)^T(x_i^k-y^k)+\frac{\mu}{2}||y^k-x_i^k||^2)+p\eta_kCD_k \qquad (37)\\
&\leq-p\eta_k(f(y^k)-f(x^*)+\frac{\mu}{2}||y^k-x^*||^2)+2p\eta_kCD_k,
\end{aligned}$$

where $\bar{g}^k\in\partial f_i(y^k)$. In (36) and (37), we use the following property of strong convexity ((B.3) of [32])

$$f(y)\geq f(x)+g^T(y-x)+\frac{\mu}{2}||y-x||^2,$$

where $g\in\partial f(x)$. The last inequality follows from $-(\bar{g}^k)^T(x_i^k-y^k)\leq CD_k$ and $||x_i^k-x^*||^2+||y^k-x_i^k||^2\geq||y^k-x^*||^2$.
Plugging it into (23), we have

$$\begin{aligned}
&\mathbb{E}[||y^{k+1}-x^*||^2|\{x_i^k\}_{i=1}^n]\\
&\leq(1-p\mu\eta_k)||y^k-x^*||^2+\frac{4\eta_k^2(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}\\
&\quad+6p\eta_kCD_k-2p\eta_k(f(y^k)-f(x^*))\\
&=(1-p\mu\eta_k)||y^k-x^*||^2+\frac{4\eta_k^2(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}\\
&\quad+6p\eta_kCD_k-2p\eta_k(f(x_j^k)-f(x^*)+f(y^k)-f(x_j^k))\\
&\leq(1-p\mu\eta_k)||y^k-x^*||^2+\frac{4\eta_k^2(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}\\
&\quad+8p\eta_kCD_k-2p\eta_k(f(x_j^k)-f(x^*))
\end{aligned}$$

for any j. Taking expectations over $\{x_i^k\}_{i=1}^n$ and plugging the bound of $\mathbb{E}D_k$, we have

$$\begin{aligned}
&\mathbb{E}(f(x_j^k)-f(x^*))\\
&\leq\frac{1}{2p\eta_k}[(1-p\mu\eta_k)\mathbb{E}||y^k-x^*||^2-\mathbb{E}||y^{k+1}-x^*||^2]\\
&\quad+\frac{2\eta_k((1-p)+np+\frac{1}{2}n)C^2}{n}+4npBC^2\sum_{r=1}^k\eta_{r-1}\theta^{k-r}.
\end{aligned}$$

Multiplying both sides by k and telescoping from $k=1$ to K yields

$$\begin{aligned}
&\sum_{k=1}^K k\mathbb{E}(f(x_j^k)-f(x^*))\\
&\leq\frac{1-p\mu\eta_1}{2p\eta_1}\mathbb{E}||y^1-x^*||^2-\frac{K}{2p\eta_K}\mathbb{E}||y^{K+1}-x^*||^2\\
&\quad+\sum_{k=2}^K(\frac{k}{2p\eta_k}(1-p\mu\eta_k)-\frac{k-1}{2p\eta_{k-1}})\mathbb{E}||y^k-x^*||^2\\
&\quad+\sum_{k=1}^K\frac{2k\eta_k((1-p)+np+\frac{1}{2}n)C^2}{n}\\
&\quad+\sum_{k=1}^K4knpBC^2\sum_{r=1}^k\eta_{r-1}\theta^{k-r}.
\end{aligned}$$

Letting $\eta_k=\frac{2}{\mu p(k+1)}$, we have $\frac{k}{2p\eta_k}(1-p\mu\eta_k)=\frac{k-1}{2p\eta_{k-1}}$. Then for the last term, we have

$$\begin{aligned}
\sum_{k=1}^K k\sum_{r=1}^k\eta_{r-1}\theta^{k-r}&=\sum_{k=1}^K k\sum_{z=0}^{k-2}\eta_{k-z-1}\theta^z\\
&=\sum_{z=0}^{K-2}\theta^z\sum_{k=z+1}^K k\eta_{k-z-1}\\
&\leq\frac{1}{1-\theta}\sum_{k=1}^K k\eta_{k-1}\\
&=\frac{1}{1-\theta}+\frac{2(K-1)}{(1-\theta)\mu p}.
\end{aligned}$$

Then

$$\begin{aligned}
&\sum_{k=1}^K k\mathbb{E}(f(x_j^k)-f(x^*))\\
&\leq-\frac{\mu K(K+1)}{4}\mathbb{E}||y^{K+1}-x^*||^2+\frac{4K(1-p+np+\frac{1}{2}n)C^2}{n\mu p}\\
&+\frac{8(K-1)nBC^2}{(1-\theta)\mu}+\frac{4npBC^2}{1-\theta}.
\end{aligned}$$

Multiplying both sides by $\frac{2}{K(K+1)}$, we have

$$\begin{aligned}
&\frac{2}{K(K+1)}\sum_{k=1}^K k\mathbb{E}(f(x_j^k)-f(x^*))\\
&\leq\frac{1}{K+1}(\frac{8(1-p+np+\frac{1}{2}n)C^2}{n\mu p}+\frac{16nBC^2}{(1-\theta)\mu}+\frac{8npBC^2}{K(1-\theta)}).
\end{aligned}$$

Since f is convex, we have the final result.

APPENDIX C
PROOF OF THEOREM 3

We still prove the theorem based on (23). For the last term of (23), starting from (25) we have

$$\begin{aligned}
&-\frac{p\eta_k}{n}\sum_{i=1}^{n}(g_i^k)^T(y^k-x^*)\\
&\le -\frac{p\eta_k}{n}\sum_{i=1}^{n}(f_i(x_i^k)-f_i(x^*))+p\eta_k CD_k\\
&\le -\frac{p\eta_k}{n}\sum_{i=1}^{n}(f_i(y^k)-f_i(x^*))+2p\eta_k CD_k\\
&= -p\eta_k(f(y^k)-f(x^*))+2p\eta_k CD_k.
\end{aligned}$$

Meanwhile since f is μ-strongly convex, L-smooth, and $\nabla f(x^*)=0$, we have (Section B.1 of [32])

$$\frac{\mu}{2}||y-x^*||^2 \le f(y)-f(x^*) \le \frac{L}{2}||y-x^*||^2.$$

So (23) can be transformed into

$$\begin{aligned}
&\frac{2}{L}\mathbb{E}[f(y^{k+1})-f(x^*)|\{x_i^k\}_{i=1}^n]\\
&\le (\frac{2}{\mu}-2p\eta_k)(f(y^k)-f(x^*))\\
&+\frac{4\eta_k^2(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}+6p\eta_k CD_k. \qquad (38)
\end{aligned}$$

Taking expectations over $\{x_i^k\}_{i=1}^n$ and plugging the bound of $\mathbb{E}D_k$ into (38), we have

$$\begin{aligned}
&\mathbb{E}[f(y^{k+1})-f(x^*)] \le (\frac{L}{\mu}-pL\eta_k)\mathbb{E}(f(y^k)-f(x^*))\\
&+\frac{2\eta_k^2 L(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}\\
&+3np^2LBC^2\eta_k\sum_{r=1}^{k}\eta_{r-1}\theta^{k-r}. \qquad (39)
\end{aligned}$$

When $\eta_k=\eta$ is fixed, then

$$\begin{aligned}
&3np^2LBC^2\eta_k\sum_{r=1}^{k}\eta_{r-1}\theta^{k-r}\\
&\le \frac{3np^2LBC^2\eta^2}{1-\theta}+3np^2LBC^2\eta\theta^{k-1} \le \frac{3np^2LBC^2(\eta^2+\eta)}{1-\theta},
\end{aligned}$$

since $\theta^{k-1}\le\frac{1}{1-\theta}$. Define

$$R=\frac{\frac{2\eta^2L(p(1-p)+np^2+\frac{1}{2}np)C^2}{n}+\frac{3np^2LBC^2(\eta^2+\eta)}{1-\theta}}{1-\frac{L}{\mu}+pL\eta}.$$

Now we can write (39) as

$$\begin{aligned}
&\mathbb{E}[f(y^{k+1})-f(x^*)]-R\\
&\le (\frac{L}{\mu}-pL\eta)(\mathbb{E}(f(y^k)-f(x^*))-R)\\
&\le (\frac{L}{\mu}-pL\eta)^k(\mathbb{E}(f(y^1)-f(x^*))-R)\\
&= (\frac{L}{\mu}-pL\eta)^k(f(y^0)-f(x^*)-R)
\end{aligned}$$

since $y^1=y^0=\frac{1}{n}\sum_{i=1}^n x_i^0$. Meanwhile,

$$\begin{aligned}
&\mathbb{E}[f(x_i^{k+1})-f(x^*)]\\
&\le \mathbb{E}[f(y^{k+1})-f(x^*)]+C\mathbb{E}D_{k+1}\\
&\le \mathbb{E}[f(y^{k+1})-f(x^*)]+np\sum_{r=2}^{k}\eta C^2B\theta^{k-r}+npC^2B\theta^{k-1}.
\end{aligned}$$

If $\eta\in(\frac{L-\mu}{p\mu L},\frac{1}{p\mu})$, then $(\frac{L}{\mu}-pL\eta)\in(0,1)$. So

$$\begin{aligned}
\mathbb{E}[f(x_i^{k+1})-f(x^*)] &\le (\frac{L}{\mu}-pL\eta)^k(f(y^0)-f(x^*)-R)+R\\
&+\frac{np\eta BC^2}{1-\theta}+npC^2B\theta^{k-1}\\
&\overset{k\to\infty}{\longrightarrow} R+\frac{np\eta BC^2}{1-\theta}
\end{aligned}$$

Zai Shi received the B.E. degree from Zhejiang University, China, in 2014. He is currently pursuing the PhD degree in the Ohio State University, US. His research interests include optimization for networking and machine learning. He received university fellowship in 2016.

Atilla Eryilmaz (S'00 / M'06 / SM'17) received his M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 2001 and 2005, respectively. Between 2005 and 2007, he worked as a Postdoctoral Associate at the Laboratory for Information and Decision Systems at the Massachusetts Institute of Technology. Since 2007, he has been at The Ohio State University, where he is currently a Professor and the Graduate Studies Chair of the Electrical and Computer Engineering Department.

Dr. Eryilmaz's research interests span optimal control of stochastic networks, machine learning, optimization, and information theory. He received the NSF-CAREER Award in 2010 and two Lumley Research Awards for Research Excellence in 2010 and 2015. He is a co-author of the 2012 IEEE WiOpt Conference Best Student Paper, subsequently received the 2016 IEEE Infocom, 2017 IEEE WiOpt, 2018 IEEE WiOpt, and 2019 IEEE Infocom Best Paper Awards. He has served as: a TPC co-chair of IEEE WiOpt in 2014 and of ACM Mobihoc in 2017; an Associate Editor (AE) of IEEE/ACM Transactions on Networking between 2015 and 2019; and is an AE of IEEE Transactions on Network Science and Engineering since 2017.