Asynchronous Decentralized Optimization in Heterogeneous Systems

Michael G. Rabbat and Konstantinos I. Tsianos

Abstract—This paper studies a class of asynchronous distributed algorithms for convex optimization where nodes communicate using push-only (feedback-free) broadcast messages, and where nodes do not necessarily perform updates at the same rate. This is especially relevant for implementation in a compute cluster where resources are shared among multiple users and may be simultaneously executing multiple tasks. We consider a distributed variant of Nesterov’s dual averaging algorithm [1], making use of distributed averaging to enable a distributed implementation, and we provide conditions under which the algorithm is guaranteed to converge to the optimal solution. This requires careful scaling of the step size parameters to account for different update rates at different nodes without biasing the solution. The proposed parameter settings assume that nodes have access to readings of a global clock, but otherwise they do not need any global information, including knowledge of their own update rate nor the overall rate at which updates occur across the network. Numerical examples illustrate the performance of the proposed approach and the pitfalls of alternative step-size parameter rules.

I. INTRODUCTION

We consider the problem of decentralized convex optimization over a network. The network is composed of $N$ nodes. A private convex objective function $f_i(x) : \mathbb{R}^d \rightarrow \mathbb{R}$ is associated with each node $i = 1, \ldots, N$, and the goal of the network is to find a vector $x \in \mathbb{R}^d$ that solves

$$\begin{align*}
\text{minimize} & \quad f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x) \\
\text{subject to} & \quad x \in X
\end{align*}$$

(1)

where $X$ is a closed, non-empty, convex subset of $\mathbb{R}^d$ and the local functions $f_i$ are continuously differentiable. We consider a typical setup where node $i$ has access to a first-order oracle which returns $\nabla f_i(x)$ given an input $x \in \mathbb{R}^d$. Each node maintains its own copy $x_i$ of the optimization variable. Nodes repeatedly perform local updates using their local oracle and communicate with their neighbours.

Our primary interest is to study asynchronous algorithms operating in heterogeneous systems where nodes perform updates at different rates. Asynchronous methods using one-directional communications are most practical for distributed optimization in large-scale systems [2]. Developing methods for heterogeneous systems with performance guarantees is highly relevant when using distributed optimization methods in cluster environments (e.g., for machine learning applications). In many scenarios, nodes may require different amounts of time to query their gradient oracle for a number of reasons. Clusters are often shared environments, and some nodes may be slower due to the corresponding process receiving fewer computational cycles. Some gradients may also simply be more complex to compute than others. The probability of having such a situation increases with the network size, and synchronous algorithms are prone to significant performance degradation since then the system runs at the speed of the slowest node.

A. Related Work

During the past decade there has been tremendous interest in decentralized optimization methods, driven by a variety of applications [3]–[9]. These methods typically interleave local updates (e.g., local gradient steps) with message-passing communication. The most typical setup is where nodes pass messages to implement a distributed averaging update [10], [11], with the intuition being that the average of the local gradients $\nabla f_i(x)$ is exactly the gradient of the network-wide objective function $f(x)$. In this manner, distributed averaging serves the purpose of simultaneously spreading information across the network and synchronizing the nodes (reaching consensus on the optimal solution).

The majority of the previous work in the literature focuses on methods requiring synchronous updates and/or bidirectional message passing for distributed averaging. These assumptions are generally violated, or they are undesirable to enforce when it comes to practical implementation [2].

The push-sum protocol is an alternative approach to distributed averaging that only requires unidirectional messaging and can be implemented in an asynchronous manner. The push-sum gossip protocol was introduced in [12], although the idea can be traced back at least as far as [13], and it has since been refined and further studied in [14], [15].

A synchronous distributed optimization method, using the push-sum protocol internally, was proposed in [16]. The analysis in [16] assumes that the network remains static during the course of computation. A distributed subgradient method using the push-sum protocol is studied in [17], where the network may change at every iteration and it is assumed that nodes know their out-degree at every iteration. In addition, the method of [17] assumes that all nodes perform updates at the same rate. An asynchronous distributed subgradient method using broadcast communications is studied in [18], where it is assumed that all nodes perform local gradient updates at different rates.
updates and communicate at the same frequency. In practice, however, this may not be the case, as discussed above.

B. Contribution

This paper proposes and analyzes a distributed asynchronous method for solving separable convex optimization problems. We explicitly focus on the case where nodes do not perform updates at the same rate. The proposed method is based on a distributed version of Nesterov’s dual averaging algorithm [1], [8], and it uses push-sum messaging for distributed averaging. Appropriately setting the algorithm parameters (akin to step-sizes), is a significant issue in the asynchronous setting when nodes have heterogeneous update rates. We illustrate that incorrectly setting these parameters causes the algorithm to converge to a suboptimal value. We propose an appropriate way to set the algorithm parameters so that the algorithm converges in expectation to the optimal solution, where the randomness is with respect to the process dictating the order in which nodes perform updates. Our approach assumes that all nodes have access to a global clock, so that they are aware of what times they locally perform updates. Otherwise, no global information is required. In particular, each node does not need to know the times at which other nodes perform updates. Moreover, each node does not need to know the rates at which other nodes perform updates, nor does it need to know the average or total rate of updates across the network.

II. MODEL AND PROBLEM SETUP

A. Notation

We begin by introducing a few notational conventions used throughout this paper. For vectors \( x, y \in \mathbb{R}^d \), we work with the standard Euclidean inner product, \( \langle x, y \rangle = x^T y \), and the corresponding induced vector norm \( \|x\|_2 = \langle x, x \rangle \). We denote by \( 0_d \) (respectively \( 1_d \)) the \( d \)-dimensional vector with all entries equal to 0 (respectively 1).

B. Objective Functions

We make the following assumptions about the local objective functions \( f_i(x) \), \( i = 1, \ldots, N \) in (1).

Assumption 1: For each \( i = 1, \ldots, N \), the local objective function has the following properties:
1) (Convexity) \( f_i(x) \) is convex in \( x \);
2) (Lipschitz continuous) There exists a constant \( L > 0 \) such that for all \( x, x' \in \mathcal{X} \),
   \[ |f_i(x) - f_i(x')| \leq L \|x - x'\| \];
3) (Lipschitz continuous gradients) \( f_i(x) \) is continuously differentiable and there exists a constant \( \Gamma > 0 \) such that for all \( x, x' \in \mathcal{X} \),
   \[ \|\nabla f_i(x) - \nabla f_i(x')\| \leq \Gamma \|x - x'\| \].

C. Network Topology

In general, nodes must coordinate in order to reach a solution of the problem (1). To this end we assume that the nodes pass messages over a network whose topology is described by a directed graph \( G = (\mathcal{N}, \mathcal{E}) \) where \( \mathcal{N} = \{1, 2, \ldots, N\} \) is the set of nodes, and \( \mathcal{E} \subseteq \mathcal{N} \times \mathcal{N} \) is the set of directed edges in the network. We write \((j, i) \in \mathcal{E}\) if node \( j \) receives messages from node \( i \).

Assumption 2: We assume that \( G \) is strongly connected and that node \( i \) knows its out-neighbours,
   \[ \mathcal{N}_{out}^i = \{ j \in \mathcal{N} : (j, i) \in \mathcal{E} \} \quad . \quad (3) \]

D. Asynchronous Time Model

We study the situation where nodes update asynchronously at different rates. Following [19], we assume that each node \( i \) holds an independent Poisson clock which ticks at a rate \( \lambda_i > 0 \). For the purposes of analysis, it is convenient to think of the random sequence of events as being generated by a global Poisson clock which ticks at a rate \( \lambda = \sum_{i=1}^{N} \lambda_i \), where each clock tick is independently assigned to node \( i \) with probability \( \lambda_i/\lambda \).

Let \( t_1 < t_2 < \cdots < t_k \cdots \) denote the times at which the global clock ticks, and let \( n_k \in \mathcal{N} \) denote the node at which the \( k \)th global clock tick occurs (equivalently, the node assigned the \( k \)th global clock tick). When a node’s clock ticks, it performs a local gradient update and then sends messages to its out-neighbours.

III. ALGORITHM DESCRIPTION

We study an asynchronous version of the push-sum distributed dual averaging (PS-DDA) algorithm introduced in [2], [16]. The dual averaging algorithm sequentially updates a linear lower bound on the objective function, performing proximal projections to maintain feasibility [1]. Distributed dual averaging (DDA) is a distributed version of this method where message passing is used to perform distributed averaging, synchronizing the lower linear bounds across the network [8]. DDA is attractive because its analysis provides rates of convergence which depend explicitly on the topology of the graph \( G \) over which messages are passed. However, there are practical issues which make a direct implementation of DDA challenging, and PS-DDA overcomes these by allowing asynchronous, uni-directional communications [2]. The version of PS-DDA previously studied and analyzed in [2], [16] is synchronous, and the corresponding asynchronous analysis and implementation was missing. Here we fill that gap, providing theoretical convergence guarantees in the asynchronous setting.

We denote by \( x_i(k) \) the value of a variable \( x \) at node \( i \) at the time of the \( k \)-th global clock tick. In PS-DDA, node \( i \) maintains and updates a primal variable \( x_i(k) \in \mathbb{R}^d \), a dual variable \( z_i(k) \in \mathbb{R}^d \), and a scalar weight \( w_i(k) \in \mathbb{R} \). The variables are initialized so that \( x_i(0) = 0_d \), \( z_i(0) = 0_d \), and \( w_i(0) = 1 \).

In order to map from the dual to the primal domain, the dual averaging algorithm make use of a 1-strongly convex
proximal function $h : \mathbb{R}^d \to \mathbb{R}$, which satisfies $h(0) = 0$, $h(x) \geq 0$ for all $x \in \mathcal{X}$, and for all $x, x' \in \mathcal{X}$ and all $\theta \in [0, 1]$,

$$h(\theta x + (1 - \theta)x') \geq \theta h(x) + (1 - \theta)h(x') - \frac{\theta(1 - \theta)}{2} \|x - x'\|^2.$$ 

The corresponding projection using $h(\cdot)$ is

$$\Pi^h_{\mathcal{X}}(z, \beta) \overset{\text{def}}{=} \arg \min_{x \in \mathcal{X}} \{h(x) + \beta h(x)\}.$$ 

(4)

When working in Euclidean domains, the standard choice is $h(x) = \frac{1}{2}\|x\|^2$; see [1], [8] for a discussion of proximal functions more appropriate for other domains.

On the $k$-th global clock tick, node $n_k$ has computed the gradient $\nabla f_{n_k}(x_{n_k}(k - 1))$. This node computes a weighted version of its gradient,

$$g_{n_k}(k - 1) = \alpha_{n_k}(k - 1) \cdot \nabla f_{n_k}(x_{n_k}(k - 1)),$$

and sends messages containing the values

$$\frac{z_{n_k}(k - 1) + g_{n_k}(k - 1)}{\deg(n_k) + 1} \quad \text{and} \quad \frac{w_{n_k}(k - 1)}{\deg(n_k) + 1}$$

to its out-neighbors $j \in \mathcal{N}^{\text{out}}_{n_k}$, where $\deg(n_k) = |\mathcal{N}^{\text{out}}_{n_k}|$ denotes the out-degree of node $n_k$. Node $n_k$ updates its own variables to the values

$$z_{n_k}(k) = \frac{z_{n_k}(k - 1) + g_{n_k}(k - 1)}{\deg(n_k) + 1}$$

(5)

$$w_{n_k}(k) = \frac{w_{n_k}(k - 1)}{\deg(n_k) + 1}$$

(6)

$$x_{n_k}(k) = \Pi^h_{\mathcal{X}}\left(\frac{z_{n_k}(k)}{w_{n_k}(k)}, \beta_{n_k}(k)\right),$$

(7)

where $\{\beta_j(k)\}_{k \geq 1}$ is a sequence of non-negative, non-decreasing algorithm parameters used at node $i$, and the division in (7) is element-wise. The neighbours $j \in \mathcal{N}^{\text{out}}_{n_k}$ update their dual variables and weights upon receiving the message from $n_k$ by setting

$$z_j(k) = z_j(k - 1) + \frac{z_{n_k}(k - 1) + g_{n_k}(k - 1)}{\deg(n_k) + 1}$$

$$w_j(k) = w_j(k - 1) + \frac{w_{n_k}(k - 1)}{\deg(n_k) + 1}.$$ 

All other variables at all other nodes across the network are held fixed at their previous values.

Since our focus is on studying the effects of heterogeneous update rates (i.e., $\lambda_i \neq \lambda_j$), we disregard phenomena such as queuing and transmission delay. Specifically, we assume that when a node transmits, the messages are received by the intended recipient; there are no collisions and no dropped packets. These assumptions are reasonable if the method is run in a compute cluster using, e.g., the message passing interface (MPI) or another standard library for message passing, in which case messages are communicated over reliable TCP connections which deal with dropped packets in a manner that is transparent to the application. We also assume that updates are instantaneous; it follows from the Poisson clock model that, almost surely, no two nodes simultaneously perform updates.

Below it will be convenient to make use of vectors of indicator variables $\xi(k - 1) \in \{0, 1\}^N$, defined such that

$$\xi_i(k - 1) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } n_k = i \\ 0 & \text{otherwise.} \end{cases}$$

(8)

Let us also define

$$K_i(k) \overset{\text{def}}{=} \{k' \leq k : n_{k'} = i\}$$

(9)

to be the subset of the first $k$ global clock ticks which occur at node $i$. Let $P(k - 1)$ denote an $N \times N$ matrix with entries

$$P_{j,i}(k - 1) = \begin{cases} \frac{1}{\deg(j)} & \text{if } i = n_k \text{ and } j \in \mathcal{N}^{\text{out}}_{n_k} \cup \{n_k\} \\ 1 & \text{if } i \neq n_k \text{ and } j = i \\ 0 & \text{otherwise.} \end{cases}$$

(10)

Using the notation just introduced, the updates across the network at the $k$-th global clock tick can be expressed succinctly via the equations,

$$g_i(k - 1) = \xi_i(k - 1) \cdot \alpha_i(k - 1) \cdot \nabla f_i(x_i(k - 1))$$

(11a)

$$z_j(k) = \sum_{i=1}^N P_{j,i}(k - 1)(z_i(k - 1) + g_i(k - 1))$$

(11b)

$$w_j(k) = \sum_{i=1}^N P_{j,i}(k - 1)w_i(k - 1)$$

(11c)

$$x_j(k) = \Pi^h_{\mathcal{X}}\left(\frac{z_j(k)}{w_j(k)}, \beta_j(k)\right).$$

(11d)

Finally, node $j$ also maintains a running average, $\hat{x}_j(k) \in \mathbb{R}^d$, which it updates by setting

$$\hat{x}_j(k) = \xi_j(k - 1)\left(\frac{|K_j(k)| - 1}{|K_j(k)|}\hat{x}_j(k - 1) + x_j(k)\right) + (1 - \xi_j(k - 1))\hat{x}_j(k - 1).$$

(12)

Below we focus on determining appropriate choices for the algorithm parameters $\{\alpha_i(k)\}_{k \geq 0}$ and $\{\beta_i(k)\}_{k \geq 1}$ in order to guarantee that $f(\hat{x}_j(k))$ asymptotically converges to the optimal cost in expectation.

IV. CHALLENGES OF STEP-SIZE SELECTION IN ASYNCHRONOUS PS-DDA

To illustrate that the choice of algorithm parameters is extremely important for convergence of the method, we give an example demonstrating how a seemingly natural approach causes the method to converge to an incorrect solution. The centralized dual averaging algorithm [1] only has parameters $\beta(k)$. (The other parameters are implicitly $\alpha(k) = 1$ for all $k$.) As is standard in general first-order convex optimization, to guarantee convergence to the optimal solution one must adapt the step-size based on the number of updates performed.

Typically one takes $\beta(k) = \sqrt{k}$ for convergence of dual averaging [1]. In the asynchronous decentralized setting, a natural approach is to have each node adapt a local step-size
based on the number of updates it has performed. Suppose
node $i$ uses $\beta_i(k) = \sqrt{|K_i(k)|}$ when it updates. If nodes
update at different rates, then the step-size parameters at
different nodes will fall out of sync as time progresses; nodes
that update less frequently will use values of $\beta_i(k)$ which are
much smaller than nodes that update more frequently.

To see why this is problematic, let us make the simplifying
assumption that $\mathcal{X} \equiv \mathbb{R}^d$ (i.e., the problem is unconstrained)
and take $h(x) = \frac{1}{2} \|x\|^2$. In this case, the projection $\Pi^b_X (\cdot, \cdot)$
simplifies so that we get
\[
x = \Pi^b_X (z, \beta) = -z/\beta .
\]

In addition, suppose that $G$ is the complete graph, where
each node receives messages from all other nodes (i.e.,
$N^{\text{out}}_i = N \setminus \{i\}$ for all $i$). Then the dual variables at all
nodes (rescaled by the weights $w_i(k)$) track the network-
average dual variable, $\bar{\pi}(k) = \frac{1}{N} \sum_{i=1}^N z_i(k)$. Since $P(k)$
is column-stochastic for all $k$, it follows from (11b) that
\[
\bar{\pi}(k) = \frac{1}{N} \sum_{i=1}^N g_i(k) = \frac{1}{N} \sum_{i=1}^N \xi_i(k) \nabla f_i(x_i(k)) .
\]
Thus, the corresponding primal variable evolves as
\[
\pi(k) = \Pi^b_X (\pi(k), \beta(k)) = -\frac{1}{\beta(k)} \sum_{k'=1}^k \bar{\pi}(k') .
\]

Consequently, if $\lambda_i < \lambda_j$, then as time progresses node $j$
will contribute its gradients to $\pi(k)$ more frequently, while
node $i$ will use smaller $\beta_i(k)$, giving more overall weight to
contributions that have arrived since the last time it performed
an update. Moreover, not all nodes contribute equally to the accumulating gradient $\pi(k)$.

To illustrate this phenomenon we simulate a network of
$N = 10$ nodes arranged in a star network, with node 1 as the
“hub” and nodes 2, ..., 10 as the spokes. The local objectives
are given by
\[
f_i(x) = \|x - c_i\|^2 ,
\]
where $c_i \in \mathbb{R}^d$ is a vector known only to node $i$, and $c_i \neq c_j$.
Consequently, all nodes must cooperate in order to find
the optimal solution $x^* = \frac{1}{N} \sum_{i=1}^N c_i$. The results for two
different settings of node update rates are shown in Figure 1,
illustrating the error trajectories when nodes with different
update rates use the step-size rule $\beta_i(k) = \sqrt{|K_i(k)|}$. In
each case, one node $i'$ updates at a rate of $\lambda_{i'} = 10$ while the
others update at the rate $\lambda_i = 1$; specifically, $i' = 1$ in
Figure 1(b), and $i' = 2$ in Figure 1(c). In both cases we see
that the values converge to an incorrect solution since the
difference $f(\bar{x}_j(k)) - f(x^*)$ does not vanish. Note that
when $\lambda_i = 10$ (cf. Fig. 1(b)), node 1 updates 10 times more
than the other nodes. This drives the network to a
consensus more rapidly, while also biasing the result of the
computation to favour node 1’s local objective. When $\lambda_2 = 10$
(cf. Fig. 1(c)), node 2 performs updates more frequently,
biasing the result in favour of minimizing its local objective
disproportionately more than the others.

V. CONVERGENCE RESULT

As we have just seen, the choice of the algorithm parameters
$\alpha_i(k)$ and $\beta_i(k)$ plays a critical role in the convergence
of the asynchronous algorithm described in Section III. To
obtain an appropriate condition, we assume that all nodes
have access to a global clock. In particular, we assume that
if the $k$-th global clock tick occurs at node $n_k = i$ then node
$i$ knows the global time $t_k$. This is a reasonable assumption
for clusters of computers connected to the Internet, which are
easily synchronized using the Network Time Protocol [20].
Similarly, for networks of embedded devices, today there
exist commercially-available low-power clock chips which,
once initially synchronized, are able to maintain accurate
time for periods of time on the order of months. More
generally, we assume that when the $k$-th clock tick occurs at
node $i$ then it knows both the current global time $t_k$ as well
as the previous time its clock ticked, which we denote by
\[
\tau_i(k) = \max\{k' \in K_i(k-1)\} .
\]

In addition, we make the following assumption.

Assumption 3: The constraint set $\mathcal{X}$ is bounded with
\[
D \equiv \max_{x,x' \in \mathcal{X}} \|x - x'\| < \infty .
\]
Together with Assumption 1, this implies that
\[
\|\nabla f_i(x)\| \leq L
\]
for all $x \in \mathcal{X}$.

Our main result is the following theorem guaranteeing
that asynchronous PS-DDA converges in expectation to an
optimal solution.

Theorem 1: Let Assumptions 1–3 hold. Let $\tilde{x}_j(k)$ be the
sequence of values generated by repeating the steps (11)
and (12), with the parameters $\alpha_j(k)$ and $\beta_j(k)$ taken as
\[
\begin{align*}
\alpha_j(k) &= t_k - \tau_j(k) \\
\beta_j(k) &= \Gamma + \sqrt{t_k} ,
\end{align*}
\]
where $\Gamma$ is the gradient Lipschitz constant from Assump-
tion 1, $t_k$ is the time of the $k$-th global clock tick, and $\tau_j(k)$
is the previous time that the clock ticked at node $j$. Then for
all $j \in \mathcal{N}$,
\[
\lim_{k \to \infty} E[f(\tilde{x}_j(k))] - f(x^*) = 0
\]
where $x^* \in \arg\min_{x \in \mathcal{X}} f(x)$ is a minimizer of $f(\cdot)$ over
$\mathcal{X}$, and where the expectation is taken with respect to the
random node activation process $\xi(1), \xi(2), \ldots, \xi(k)$.

The proof of Theorem 1 is presented in [21] and is omitted
here due to space constraints.

We remark that the settings of the algorithm parameters
$\alpha_j(k)$ and $\beta_j(k)$ given in the statement of the theorem only
require that each node knows the value of the global clock
at the times when it performs an update. Nodes do not need
to know the times at which other updates occur across the
network; they do not need to know any of the update rates
$\lambda_j$ (including their own), nor do they need to be aware of
the overall update rate $\lambda$. Hence, the scheme proposed above
can be implemented in a decentralized manner.
The parameter values specified in Theorem 1 can be interpreted as follows. The choice for $\alpha_i(k)$ implies that each node scales its contribution to the average gradient by the amount of time that has passed since its last update (cf. equation (11a)). Over a sufficiently long time duration, this ensures that all nodes contribute equally to $\pi(k)$. The choice for $\beta_j(k)$ implements, in a distributed manner, the step-size rate which would be applied if $\pi(k)$ and $\pi(k)$ were begin generated at a central node. As mentioned above, this is achieved by having nodes know the time that has passed since execution of the algorithm began, without requiring any central agent or information about the update rates to coordinate the step-size values.

VI. SIMULATIONS

To illustrate the performance of asynchronous PS-DDA with the parameters $\alpha_i(k)$ and $\beta_i(k)$ prescribed by Theorem 1, we revisit the example from Figure 1. Recall that we have a star network with $N = 10$ nodes. In the first scenario, the update rate at node 1 is $\lambda_1 = 10$ and the rates at all other nodes are $\lambda_i = 1$. In the second scenario, the update rate at node 2 is $\lambda_2 = 10$, and the rates at other nodes are $\lambda_i = 1$. The performance of PS-DDA with appropriate parameters is shown in Figure 2. As suggested by Theorem 1, the optimality gap $f(\hat{x}_i(k)) - f(x^*)$ converges to zero for all nodes $i = 1, \ldots, N$, regardless of the rates $\lambda_i$.

In the example above, one node (the one with $\lambda_1 = 10$) updates at a faster rate than the rest of the network. As seen by comparing Fig. 1 and Fig. 2, if not properly accounted for this can cause the algorithm to converge to an erroneous solution. To illustrate why asynchronous updates may lead to faster convergence than synchronous ones consider an alternative example where one node updates at a slower rate than the rest. Specifically, for the same star network with 10 nodes, suppose that $\lambda_2 = 0.5$ and $\lambda_i = 1$ for all other nodes. One could adopt a synchronous approach where all nodes wait until they have received messages from all of their neighbours before proceeding to the next iteration, such as in the synchronous PS-DDA algorithm [16]. The drawback of this approach is that all nodes proceed at the pace of the slowest node. For the particular example considered here, the resulting convergence behaviour is shown in Figure 3, where we plot the node-averaged error, $\frac{1}{N} \sum_{i=1}^{N} f_i(\hat{x}_i(k)) - f(x^*)$ versus time for both the synchronous and asynchronous approaches. The rate at which the synchronous algorithm approaches the optimal solution is slowed down by the update rate of node 2.

VII. CONCLUSIONS

Practical applications of large scale distributed optimization require algorithms that are flexible and run reliably even when the nodes have fluctuations and variations in their speed of computation. Of the large body of recent work on distributed optimization, very few come with convergence guarantees when executed in heterogeneous asynchronous environments. In this work, we bridge the gap for consensus-based optimization by describing an asynchronous decentralized algorithm called PS-DDA which has provable convergence guarantees. We discuss the delicate issues of modelling asynchronism and selecting appropriate step sizes, and we prove convergence under arbitrarily different per-node update rates.

The main theoretical result for asynchronous PS-DDA guarantees convergence in expectation. With a little extra work we believe that the same method can be shown to have stronger convergence guarantees (in probability and almost surely). In order to study systems with heterogeneous update rates we assumed that the times when nodes perform updates follow a Poisson process. This simplifies portions of the analysis but is not essential, and our current work involves extending the results presented here to general ergodic update processes, including those with time-varying per-node update rates. Appropriately setting the algorithm parameters requires that nodes have access to a global clock. Given the stochastic nature of the updates and analysis, it is straightforward to extend the analysis to handle the case where nodes make noisy observations the times $t_k$ at which their local clock...
Asynchronous ticks (e.g., corrupted with zero-mean additive noise), as long as the noise is bounded.

REFERENCES


