Consensus-Based Distributed Optimization: Practical Issues and Applications in Large-Scale Machine Learning

Konstantinos I. Tsianos, Sean Lawlor, and Michael G. Rabbat
Department of Electrical and Computer Engineering
McGill University, Montréal, Québec, Canada
Email: konstantinos.tsianos@mail.mcgill.ca, sean.lawlor@mail.mcgill.ca, michael.rabbat@mcgill.ca

***Abstract*— This paper discusses practical consensus-based distributed optimization algorithms. In consensus-based optimization algorithms, nodes interleave local gradient descent steps with consensus iterations. Gradient steps drive the solution to a minimizer, while the consensus iterations synchronize the values so that all nodes converge to a network-wide optimum when the objective is convex and separable. The consensus update requires communication. If communication is synchronous and nodes wait to receive one message from each of their neighbors before updating then progress is limited by the slowest node. To be robust to failing or stalling nodes, asynchronous communications should be used. Asynchronous protocols using bi-directional communications cause deadlock, and so one-directional protocols are necessary. However, with one-directional asynchronous protocols it is no longer possible to guarantee the consensus matrix is doubly stochastic. At the same time it is essential that the coordination protocol achieve consensus on the average to avoid biasing the optimization objective. We report on experiments running Push-Sum Distributed Dual Averaging for convex optimization in a MPI cluster. The experiments illustrate the benefits of using asynchronous consensus-based distributed optimization when some nodes are unreliable and may fail or when messages experience time-varying delays.**

I. Introduction

Consensus-based optimization algorithms have the appealing feature that they can operate in a peer-to-peer fashion, with minimal coordination between nodes. Much of the existing literature has focused on establishing and analyzing convergence properties of these algorithms. This paper discusses issues arising when implementing and using consensus-based algorithms for distributed optimization in practice, complementing the existing literature. We find that having asynchronous algorithms which use one-directional (push-based) communications and which do not rely on doubly-stochastic consensus parameters are the most desirable from the perspective of developing a robust and efficient implementation.

The dramatic increase in available data has made imperative the use of parallel and distributed algorithms for solving large-scale optimization and machine learning problems (see for example [1], [2]). Among numerous alternatives, a significant amount of research has focused on developing consensus-based algorithms [2]–[7] which combine some version of local optimization with a distributed consensus algorithm running over a peer-to-peer type of network. These methods typically focus on solving separable minimization problems of the form

$$\text{minimize} \quad F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{1}$$

$$\text{subject to} \quad x \in \mathcal{X} \tag{2}$$

where $\mathcal{X}$ is a convex constraint set, and it is typically assumed that the functions $f_i(x)$ are Lipschitz continuous, possibly with Lipschitz continuous gradient. The standard distributed setup involves a network of n nodes, and the objective $f_i(x)$ is only known to node i. For example, evaluating $f_i(x)$ or its gradient may require access to a large amount of data (e.g., if the objective is to minimize loss over a very large distributed dataset), in which case it is not practical to exchange data. Instead, the nodes must coordinate and send other information such as primal variables $x_i(t)$ or gradients of $f_i(x)$ evaluated at a particular value. In distributed optimization, this communication occurs in an unstructured manner over a topology represented as a graph G; that is, nodes exchange messages with their neighbors in G. The communication topology G is always assumed to be connected so that no nodes are isolated from the rest.

With such an approach, all computing nodes have the same role in the optimization procedure, thereby eliminating single points of failure and increasing robustness. This is important in large-scale systems where machines may fail or stall during the computation. At the same time, consensus-based algorithms are simple to implement and avoid the overhead involved in other approaches (e.g., aggregating up a spanning tree), especially when the network topology is time-varying. One downside of consensus algorithms is that peer-to-peer networks lack a highly organized infrastructure, and coordinating the computing nodes becomes a challenge. Consensus algorithms have been analyzed mostly in wireless network settings [8] and provide an elegant distributed solution for coordinating the nodes. In this article we consider implementations of consensus-based distributed optimization algorithms in wired networks.

In this article, we concentrate on the role and implications of network constraints on distributed consensus and consensus-based distributed optimization. We provide a unified view of how the network affects both the speed of convergence as well as the solution to which the algorithm converges. We argue that averaging, one-directional communication and asynchronism are key aspects that a consensus algorithm should posses to be of practical importance. To accommodate all three aspects, it is necessary to relinquish some of the most popular algorithms that rely on the elegant properties of doubly stochastic matrices. We conclude the paper with experiments that complement the theoretical discussion and illustrate the effects of the network in practice.

II. Previous Work

The main motivation for this work comes from applications in consensus-based distributed optimization [9], [10] which has recently received a lot of attention in the context of large-scale optimization and machine learning, as well as in wireless sensor networks. Consensus-based distributed optimization algorithms generally interleave a local gradient descent step with an iteration of distributed averaging to coordinate or synchronize values across the network

The general theme of these algorithms is the interleaving of a local optimization routine with a distributed consensus algorithm to coordinate the nodes of a network towards the optimum of a separable function whose components are distributed over the compute nodes. The recent literature includes many algorithms that exchange either the state information [2], [4]–[6], [11] or dual information [2], or gradient information [3], [12]. The focus is mostly on showing convergence for constrained or unconstrained problems at a rate which depends on properties of the convex objective such as smoothness or L-Lipchitz continuity. Very little attention has been put in practical and implementation aspects.

Integral part of the aforementioned distributed optimization algorithms are the consensus algorithms for which the literature is quite rich. See [8], [13] and references there in. A lot of effort has been put into analyzing the rate of convergence to consensus [14]. Very important in that direction has been the connection of consensus algorithms to the convergence or Markov chains [15]. Specifically, the Markov chain view reveals the dependence of the convergence speed on the spectral properties of the underlying network. Of practical interest is the case of asynchronous consensus algorithms and the closely related case of time varying networks or consensus matrices. In [16] is it shown that using asynchronous broadcasts and forming convex combinations of incoming information guarantees convergence to the average only in expectation. If we model time varying networks, [17] provides necessary conditions under which convergence is achieved while [18] characterizes the expectation and variance of the consensus value. Interestingly, using a different type of algorithm called Push-Sum [19], [20] convergence to the true average under the same conditions for time varying networks or consensus matrices is guaranteed.

III. Distributed Dual Averaging

To make the paper self-contained, we provide some necessary background on the *distributed dual averaging* (DDA) algorithm which will be used as a concrete example to illustrate the theoretical aspects in practice. For more details consult [3]. We present DDA as a prototypical example of consensus-based distributed optimization.

DDA solves the problem (1). Suppose we are given an undirected network $G = (V, E)$ of $|V| = n$ compute nodes. Each node i knows a function $f_i(x) : \mathbb{R}^d \to \mathbb{R}$. Moreover, nodes can communicate with one another using the available communication channels indicated by the edge set E. We assume that each f_i is convex and $L-Lipschitz$ continuous with respect to the same norm $\|\cdot\|$; i.e., $|f_i(x) - f_i(y)| \leq L\|x-y\|, \forall x, y \in \mathcal{X}$. As a consequence, for any $x \in \mathcal{X}$ and any subgradient $g_i \in \partial f_i(x)$ we have $\|g_i\|_* \leq L$ where $\|v\|_* = \sup_{\|u\|=1}\langle u, v\rangle$ is the dual norm.

Let us select select a 1-strongly convex function $\psi : \mathbb{R}^d \to \mathbb{R}$ such that $\psi(x) \geq 0$ and $\psi(0) = 0$. For example, take $\psi(x) = \|x\|_2^2$. Also select a non-increasing sequence of positive step sizes $\{a(t)\}_{t=0}^{\infty}$ and a doubly stochastic matrix $P = [p_{ij}]$ that respects the structure of G in the sense that $p_{ij} > 0$ only if $i = j$ or $(i, j) \in E$. DDA proceeds as follows. Each node maintains a primal variable $x_i(t)$ and a dual variable $z_i(t)$. At iteration t, node i performs two steps:

1. Communicate: Send $z_i(t)$ to and receive $z_j(t)$ from neighbors.

2. Compute: Update the primal and dual variables by setting

$$z_i(t+1) = \sum_{j=1}^{n} p_{ij} z_j(t) - g_i(t) \tag{3}$$

$$x_i(t+1) = \Pi_{\mathcal{X}}^{\psi}\left(z_i(t+1), a(t)\right) \tag{4}$$

where $g_i(t) \in \partial f_i(x_i(t))$ and the projection operator $\Pi_{\mathcal{X}}^{\psi}(\cdot,\cdot)$ is defined as

$$\Pi_{\mathcal{X}}^{\psi}\left(z, a\right) = \underset{x\in\mathcal{X}}{\operatorname{argmin}}\{\langle z, x\rangle + \frac{1}{a}\psi(x)\}. \tag{5}$$

In [3] it is shown that, for the updates above, the local running average $\hat{x}_i(T) = \frac{1}{T}\sum_{t=1}^{T} x_i(t)$ converges to the optimum at a rate $O(\frac{\log(\sqrt{n}T)}{\sqrt{T}})$. Specifically, keeping track of the average cumulative gradient

$$\bar{z}(t) = \frac{1}{n}\sum_{i=1}^{n} z_i(t), \tag{6}$$

the following basic theorem is proven.

Theorem 1 ([3]): Let the sequences $\{x_i(t)\}_{t=0}^{\infty}$ and $\{z_i(t)\}_{t=0}^{\infty}$ be generated by the updates (3),(4) using a non-increasing step size sequence $\{a(t)\}_{t=0}^{\infty}$. For any $x^* \in \mathcal{X}$ and for every node $i \in V$ we have:

$$\begin{aligned} f(\hat{x}_i(T)) - f(x^*) \leq & \frac{1}{Ta(T)}\psi(x^*) + \frac{L^2}{2T}a(t-1) \\ & + \frac{2L}{nT}\sum_{t=1}^{T}\sum_{j=1}^{n} a(t)\left\|\bar{z}(t) - z_j(t)\right\|_* \\ & + \frac{L}{T}\sum_{t=1}^{T} a(t)\left\|\bar{z}(t) - z_i(t)\right\|_*. \end{aligned} \tag{7}$$

The first two terms in (7) are common in subgradient optimization algorithms while the last two terms capture the network error due to the discrepancy between the local gradients and the true average gradient. By bounding the network error $\|\bar{z}(t) - z_i(t)\|_*$, we can derive convergence rates that depend on the network characteristics.

DDA interleaves a local gradient step with a consensus step in (3), and this involves communication. It should be clear that the underlying communication network has an effect on the performance of the optimization algorithm. From a theoretical standpoint, the main factor is node connectivity for quick information diffusion. From a more practical view, possible communication delays and also the actual communication exchange mechanism (e.g., blocking vs non-blocking communication) play an important role. Notice that, as described above, DDA is a synchronous algorithm; i.e., all nodes exchange information and *then* perform the updates (4) and (4) simultaneously. As we will see below, this may be difficult or even undesirable to enforce in an actual implementation.

IV. Distributed Consensus Algorithms

Recall equation (3) of the DDA algorithm. If we disregard for the moment the addition of the latest local gradient $g_i(t)$, we see that a consensus step involving communication with other nodes is involved. Each node (i) receives the variables $z_j(t)$ from its neighbors and forms a convex combination. Intuitively, for the network to agree on the solution with minimizes $f(x)$, the nodes need to agree on the direction to the optimal value which is locally captured by each variable z_i. This notion of agreement, or consensus in a network is described as follows.

Assume each node i in a network G holds a value z_i. We stack the initial values in a vector $\boldsymbol{z}(0) = (z_1, \dots, z_n)^T$. The general consensus problem asks for a distributed algorithm such that the nodes of the network iteratively exchange messages with their neighbours in order to reach consensus; i.e., $\boldsymbol{z}(t) \to c\mathbf{1}$ as $t \to \infty$. In other words, we want the nodes to agree on a common value c using only local communication with neighboring nodes. The consensus literature is quite voluminous but most of the proposed schemes revolve around a linear iteration scheme of the form

$$\boldsymbol{z}(t) = P(t)\boldsymbol{z}(t-1). \tag{8}$$

It follows from Perron-Frobenius theory that if we choose a time-homogeneous row stochastic matrix $P(t) \equiv P \in \mathbb{R}^{n \times n}$ such that $P\mathbf{1} = \mathbf{1}$ and $p_{ij} > 0$ if $(j, i) \in E$, consensus is achieved almost surely on a value c that is a convex combination of the initial values $\boldsymbol{z}(0)$.

In the following subsections, we analyze the consensus iteration from a practical standpoint. We first distinguish certain properties that a theoretical scheme should posses to be applicable. This restricts the choices of permissible P matrices and, in turn, limits the available theoretical tools that can be applied to analyze consensus. The last part of this section discusses theoretical results that may still be useful.

A. Average Consensus

A very popular special case is the average consensus problem where the limit value must be the average of the initial values; i.e., $c = \frac{1}{n}\sum_{i=1}^n z_i(0)$. As it turns out, for the purposes of consensus-based distributed optimization, an averaging protocol is necessary in order not to bias the objective function being optimized. The importance of averaging has been mentioned in previous work on distributed optimization (e.g., [4], [21]). To see the reason, consider equation (3). Unwrapping the recursion and assuming zero initial conditions for simplicity, we have

$$z_i(t) = -\sum_{s=1}^{t-1}\sum_{j=1}^{n} \left[P^{t-s-1}\right]_{ij} g_j(s) - g_i(t). \tag{9}$$

As t grows and P^{t-s-1} converges to its limit $\mathbf{1} \cdot \boldsymbol{\pi}^T$ where $\boldsymbol{\pi}$ is the stationary distribution of P, the gradients of different nodes are weighted based on the stationary distribution $\boldsymbol{\pi}$, and this weighting is unequal unless $\boldsymbol{\pi}$ is the uniform distribution. The implication for consensus-based optimization is that instead of minimizing the true objective (1) a naïve consensus-based approach will minimize the biased objective $\tilde{f}(x) = \sum_{i=1}^n \pi_i f_i(x)$.

To avoid this problem, most previous work has insisted of sticking to doubly stochastic protocols P, i.e., protocols where $\mathbf{1}^T P = \mathbf{1}^T$. Such protocols are, by definition, averaging protocols. As we will explain below however, doubly stochastic protocols are undesirable to use in practice because they require synchronization and coordination. Furthermore, it turns out that averaging can be achieved without them. For example, [21] shows that a simple reweighing of the objective removes the bias and achieves averaging for any row-stochastic matrix P with stationary distribution $\boldsymbol{\pi}$ (see also the *scaled agreement algorithm* in [22]):

$$f(x) = \frac{1}{n}\sum_{i=1}^n f_i(x) = \sum_{i=1}^n \pi_i \left[\frac{f_i(x)}{\pi_i n}\right] = \sum_{i=1}^n \pi_i h_i(x). \tag{10}$$

Note that an implementation of this approach requires that both $\boldsymbol{\pi}$ and n are known. Moreover, it requires that P be time-homogeneous, which is not reasonable when the network induces time-varying delays.

A different solution is proposed in [12], [23] by employing an algorithm called Push-Sum [19]. This approach makes use of a column-stochastic matrix P that is compatible with the structure of network as embodied in G. In addition to the parameters $z_i(t)$, each node also maintains a weight variable $w_i(t)$ which is initialized to $z_i(0) = 1$. Then, by repeating iteration (8) together with an iteration $\boldsymbol{w}(t+1) = P\boldsymbol{w}(t)$, it can be shown that $\frac{z_i(t)}{w_i(t)}$ approaches the average as $t \to \infty$. This is a very elegant solution, since the nodes do not need to know the stationary distribution of P (they are implicitly computing it through the iteration on the weights) or the network size. Furthermore, Push-Sum works even when the weights $P(t)$ vary with time [20].

B. One-Directional Communication

There exist consensus algorithms with bi-directional communication between nodes. For example, a popular algorithm for distributed averaging is Randomized Gossip [15]. This is a very simple asynchronous protocol where at each iteration a pair of neighbours exchange values and set their new values to the pairwise average of their previous values. In the context of distributed optimization, this protocol is suggested in [3] as a way to reduce the communication overhead when nodes have a large number of neighbours. However, in practice, the bi-directional communication model can be problematic as it creates deadlocks. Consider, for example, three nodes i, j, and k connected in a clique. Without coordination, there is no way of enforcing a rule that only two neighbours activate in one time instant and that no other node initiates an averaging update until the one currently in progress is completed. Consequently, suppose i attempts to exchange its value with j. While i blocks, expecting j's response, node j attempts to exchange values with k. While j is blocked waiting, k, being oblivious to the situation, independently initiates an exchange with i. Now there is a deadlock since i cannot proceed without j's value, j cannot proceed without k's value, and k is waiting for i. Researchers have previously argued that (theoretically) this will not happen if the time it takes to transmit information is much shorter than the frequency of communication. However, in distributed optimization problems, where the amount of data transmitted is non-trivial, this scenario is not unlikely and we have certainly encountered it in our experiments. One fix may be to artificially slow down the rate at which nodes communicate (e.g., by sleeping for a random time before transmitting), but this is undesirable since the ultimate goal is to solve the optimization as quickly as possible. In addition, the added complexity of coordinating two nodes that exchange information via blocking receive operations or idling in non-blocking communication is usually undesirable. For these reasons, we argue that a consensus protocol must rely on one-directional communication where nodes only transmit information and then proceed with their local computations without expecting a response.

C. Semantics of Different Protocols

With the above considerations in mind, we turn our attention to protocols described by a consensus matrix $P(t)$. In discrete time, at each iteration the state vector of the node values evolves as (8). Depending on the structure and behavior of $P(t)$, such protocols can encode the semantics of one- or two-directional communication and they can drive $\boldsymbol{z}(t)$ to average consensus.

An important distinction is made depending on whether or not $P(t)$ varies with time. The case where $P(t) = P$ is time-homogenous implies that the algorithm is *synchronous*; i.e., where each node communicates with all of its neighbors in every iteration. As discussed above, synchronous protocols require that nodes block until they have received one message from each neighbor, and this is undesirable since then the entire network moves at the pace of the slowest node. Allowing the consensus matrix to be time-dependent provides more freedom to model time varying communication delays or to encode *asynchronous* communication where a node may choose whether or not to transmit something to each neighbor at each round.

To achieve average consensus with one-directional communication, previous work has insisted on using doubly stochastic consensus protocols where each row and column of $P(t)$ sums to 1. See for example [13]. However, in asynchronous and time-varying protocols, agreeing on time-varying weights that preserve double stochasticity requires additional coordination, effectively foregoing asynchronous operations. In addition, to model in the presence of communication delays, double stochasticity can be lost [23]–[25]. Finally, there might be cases where a directed network does not admit a doubly stochastic consensus matrix [26]. For these reasons we focus on the case where $P(t)$ is stochastic but not doubly stochastic.

With stochastic protocols, the nodes become disentangled and more autonomous. When $P(t)$ is row stochastic, each node controls a row of the consensus matrix (each node applies the weights in its row of $P(t)$ to the messages it receives). At each iteration, the new value at a node is a weighted average of the incoming values. The weights are encoded in the corresponding row of $P(t)$ and need to sum to 1. If on the other hand we use a column stochastic matrix $P(t)$, the semantics are different as each node controls a column of the matrix; each node sends a portion of its current value to each neighbour so that the portion fractions sum to one (as indicated by the stochastic columns).The receiver simply sums up the incoming messages which is convenient when we do not know how many messages will be received at each iteration (e.g., due to random communication delays).

D. Summary of Theoretical Results

We conclude this section by summarizing some important results and properties of (8) for the two cases of time invariant or varying consensus matrices $P(t)$. Choosing a row or column stochastic protocol has implications on what we can say about the convergence properties of (8).

As mentioned above, time-homogeneous consensus protocols, where $P(t) \equiv P$ for all t, must be implemented using synchronous, blocking communications so that each node receives a message from all of its neighbors before computing the update for each iteration. Such protocols admit fairly straightforward convergence analysis. The underlying graph G is connected by assumption. In this case, since the entries of P respect the connectivity of G (i.e., $P_{i,j} > 0$ if and only if $(i, j) \in E$), then P is an irreducible matrix. It follows that if G is symmetric and P is doubly stochastic, then it corresponds to the transition matrix of a reversible Markov chain, and a number of standard results apply. In particular, P converges to a unique stationary distribution π at a rate $O(|\lambda_2(P)|^t)$, where $\lambda_2(P)$ is the second largest eigenvalue of P (see, e.g., Theorem 4.2 in [27]). More generally, if P is not reversible (e.g., if G is not symmetric) then the theory for reversible chains can still be applied to obtain bounds by first *reversibilizing* the chain (akin to a symmetrizing transformation); see, e.g., [28]. In general, the reversibilization transforms require that P be strongly aperiodic (all diagonal elements satisfy $P_{i,i} \geq 1/2$). When this is not true, it is common to study a lazy version of the corresponding chain, $\frac{1}{2}(I+P)$. We remark that more recent results for characterizing the mixing times of non-reversible Markov chains with zero minimum holding probability [29] may lead to tighter results and this is an interesting direction for future work.

It is generally more difficult to obtain tight bounds for convergence rates of products of time-varying stochastic matrices. If the matrices $P(t)$ are drawn i.i.d. according to a known distribution, then the bounds mentioned above for time-homogeneous protocols can be applied to the expected update matrix $\mathbb{E}[P(t)]$. When all matrices $P(t)$ are row stochastic, the process (8) gives rise to a backward product, $\boldsymbol{x}(t) = P(t)P(t-1)\cdots P(1)\boldsymbol{x}(0) \overset{\text{def}}{=} T(1,t)\boldsymbol{x}(0)$. (The corresponding quantity is a forward product when the matrices are column stochastic.) Convergence properties of backward products of stochastic matrices are typically obtained by establishing weak ergodicity [27]; i.e., that $[T(r,t)]_{i,s} - [T(r,t)]_{j,s} \to 0$ as $t \to \infty$ for all i, j, s, and r, where $[T]_{i,j}$ denotes the entry of the matrix T in row i and column j. In this setting, rates of convergence are traditionally obtained using coefficients of ergodicity and related scrambling properties of the matrix process $\{P(t)\}$ [17], [27], [30]. More recently, the PhD thesis of Touri [31] provides rates of convergence for backward products using a suitable Lyapunov function and the *infinite flow property* which ensures that the graph formed by putting edges between nodes that exchange information infinitely often is connected. In general these rates of convergence are pessimistic, involving a worst-case analysis. For example, when packets can be delayed, the bounds depend on the largest possible delay.

V. Push-Sum Distributed Dual Averaging

Based on the discussion in the previous section, Distributed Dual Averaging is not a practical algorithm. It is a synchronous algorithm that relies on a doubly stochastic consensus protocol P. DDA can however be used as a base for constructing a new distributed optimization algorithm that does not have those shortcomings. Push-Sum Distributed Dual Averaging (PS-DDA) is presented and analyzed in [12]. The fundamental difference is in the consensus algorithm used which, as the name, suggests is the Push Sum algorithm.

A. Asynchronous Distributed Optimization

The algorithm introduced in [12] is a variation on distributed dual averaging to make it suitable for implementation in a cluster. In particular, it uses asynchronous updates, one-directional push-only communications, and it does not require that the consensus matrix P be doubly stochastic; rather P should be column stochastic. PS-DDA works similar to distributed dual averaging. In addition to the primal and dual variables $x_i(t)$ and $z_i(t)$, each node i also maintains a weight $w_i(t)$. At each iteration, node i transmits messages with $p_{ji}w_i(t)$ and $p_{ji}z_i(t)$ to each neighbor j. Then it computes updates

$$w_i(t+1) = \sum_{j=1}^{n} p_{ij} w_j(t) \tag{11}$$

$$z_i(t+1) = \sum_{j=1}^{n} p_{ij} z_j(t) - g_i(t) \tag{12}$$

$$x_i(t+1) = \Pi_{\mathcal{X}}^{\psi}\left(\frac{z_i(t+1)}{w_i(t+1)}, a(t)\right). \tag{13}$$

The weight variable $w_i(t)$ automatically rescales the dual variables to account for the case where the stationary distribution of P is non-uniform. In this way, PS-DDA more naturally accommodates the challenges posed by a real implementation. However, as we discuss next, there still exist implementation subtleties that need to be addressed.

B. Implementation Remarks

Here we summarize a number of issues that arise and must be addressed in a real implementation. We assume that each node has the ability to send and receive messages and also to poll its buffer for incoming messages that have not been received yet. In our implementation, discussed in Section VI below, these features are provided by the *message passing interface* (MPI) library.

1) One directional communication: PS-DDA uses one-directional communications so that each node sends information without expecting replies before it updates. Specifically, node i rescales $w_i(t)$ and $z_i(t)$ by p_{ji} and send the rescaled values to its neighbor j. The receiver forms the sum of the receives w and z messages.

2) Numerical instability: In asynchronous mode, each node independently decides when to send a message to its neighbours. Moreover, a message w from node i to node j is $p_{ji}w_i(t)$. If, for some reason, node i finishes its iterations faster than its neighbours (in which case its incoming message buffer will be empty most of the time and so it will not update (11)), it is possible that $w_i(t)$ will become very small due to repeated rescaling by $p_{ji} < 1$. In practice after a few thousand iterations we could hit the numerical precision limits. To prevent this from happening, it is sufficient to add a condition that prevents node i from transmitting if $w_i(t)$ is too small. We use a threshold of 10^{-40} in our experiments and find that this value suffices to avoid any numerical instabilities.

3) Step-size de-synchronization: By allowing the consensus matrix P to be time-varying the algorithm becomes asynchronous. Note, however, that the description (11)-(13) is in terms of iterations and the step size at each node is set as $a(t) = O\left(\frac{1}{\sqrt{t}}\right)$. If each node operates at its own pace, maintaining a local iteration counter, the step sizes can end up being very different at different nodes. This situation can be problematic in practice because the incoming messages will be discounted by step sizes that differ by orders of magnitude at different nodes just because the nodes are at different stages of the computation. To prevent this from happening, we update the step size based on actual wall clock time instead of iterations. Each node maintains a secondary iteration counter τ which is incremented by 1 every 100ms. This way if a node experiences a delay and finishes an iteration in, e.g., 1 second, then it will set $\tau^{\text{new}} = \tau^{\text{old}} + 10$ and $a(\tau^{\text{new}}) = \frac{1}{\sqrt{\tau^{\text{new}}}}$. In a somewhat controlled environment like a cluster, where nodes begin the computation at effectively the same time, this solution suffices. A theoretical analysis of the effect of de-synchronized step-sizes is an important topic for future work.

4) Incoming message handler: Clusters are shared resources, and it is not uncommon for one node to be simultaneously assigned to process multiple tasks for different users. Moreover, network throughput may vary significantly depending on other background traffic. Both of those factors can result in some nodes transmitting more frequently than others. Consequently, when the (slow) receiver polls its incoming message buffer it may find multiple messages from the same neighbor. Those messages would have been sent at different moments in time, but they arrive and are processed during the same update due to, e.g., communication delays. It is an interesting question how to handle such incoming information. At the one extreme, a node can wait until it receives at least one message per neighbour. We have found that this approach does not work well in practice and goes against the desired asynchronous operation. At the other extreme, a node may empty its incoming buffer and sum all the incoming messages at the beginning of each iteration. Then the sums in (11) and (12) are over all messages in the buffer, not over all nodes. The danger in this case is to run into a producer-consumer scenario where one node continuously sends new information while another node continuously receives it. In this scenario, the receiver may never exit the receive mode to continue with local computations. Although this could happen in principle, in practice we did not observe this behaviour. It would be interesting to explore if assigning less weight to older messages can speedup convergence.

5) Communicator saturation: Depending on the CPU and the problem being solved, it is possible that the local gradient and projection computations will be very quick, and a node may poll its communicator (at the lowest level a TCP socket) very frequently for new messages. This is not uncommon in practice and can result in the system stalling simply because the network cannot be polled too frequently. This issue can also be easily prevented by making sure that a minimal amount of time always lapses between polls. A value of 10ms was found be sufficient in our implementation.

VI. Experimental Evaluation

A. Benchmark Problem and Setup

To complement the discussion above, report experiments using implementations of DDA and PS-DDA on a cluster of 15 nodes. Each node has a 3.2 GHz Pentium 4HT processor and 1 GB of memory, and they are physically connected in a star topology through an Ethernet switch that allows for roughly 11 MB/sec throughput per node. Our implementation is in C++ using the send and receive functions of OpenMPI v1.4.4 for communication. The Armadillo v2.3.91 library, linked to LAPACK and BLAS, is used for efficient numerical computations.

As a benchmark problem we seek to minimize a sum of quadratics with

$$f_i(x) = \sum_{j=1}^{M} (x - c_{j|i})^T (x - c_{j|i}) \tag{14}$$

where $x \in \mathbb{R}^{5,000}$, $M = 500$ and $c_{j|i}$ is the centre of the j-th quadratic of node i. The $c_{j|i}$ are designed so that the minima of the components $f_i(x)$ at each node are very different, so that coordination is essential in order to obtain an accurate optimizer of $F(x)$.

B. There Exist No Doubly Stochastic Matrices in Practice

The first experiment aims to illustrate that it is not possible to guarantee that the updates $P(t)$ are doubly stochastic due to network delays and nodes that experience different amounts of communication overhead and workload, even if the consensus protocol is initially designed to be doubly stochastic. Consequently, the standard consensus algorithm is not an averaging algorithm anymore and convergence to the right solution is lost.

To illustrate the point, we solve problem (14) on a 15-node with neighborhood structure defined through P so that node 1 has a much higher degree than all other nodes, as shown in Figure 1. We expect that node 1 will spend more time communicating than the others, and its iterations will take more time to complete. We select $P = I - \frac{D-A}{d_{max}+1}$ where, D is a diagonal matrix containing the node degrees (excluding self loops), A is the symmetric graph adjacency matrix and d_{max} is the maximum node degree. It can be verified that P is doubly stochastic. Figure 2 shows the evolution of the objective value $F(x_i(t))$ at each node when we solve the problem using DDA (i.e., with asynchronous consensus updates that are not doubly stochastic, not asynchronous Push-Sum). Note that if the true consensus matrix used at each iteration was indeed the doubly stochastic matrix P, then all nodes would converge to the average consensus solution. However, this is clearly not the case in practice. As the figure shows, node 1 being slower than the rest, cannot coordinate with the team. The resulting consensus protocol is no longer averaging and we have disagreement. On the other hand, if we make use of the asynchronous Push-Sum weights and run PS-DDA, then as Figure 3 shows consensus does work and the objective is minimized as desired.

C. Comparison with AllToAll

In the next experiment, we compare the performance of consensus-based PS-DDA with a solution that uses MPI's specialized All2All communication capabilities. The latter is available in high-performance computing clusters supporting MPI and allows for all the nodes to exchange information with each other and obtain the true average z at each iteration. All2All is designed as an efficient primitive to allow all nodes in the network to exchange information in

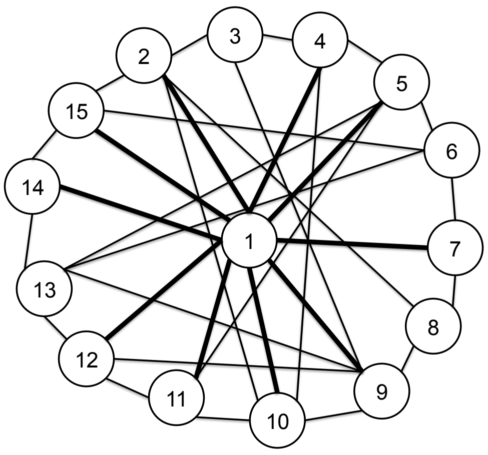

Fig. 1. The unbalanced communication topology used in the first set of experiments. In this topology, node 1 has many more neighbors than the others, and consequently, it spends more time communicating than other nodes. Edges connected to node 1 have a heavier weight than other edges.

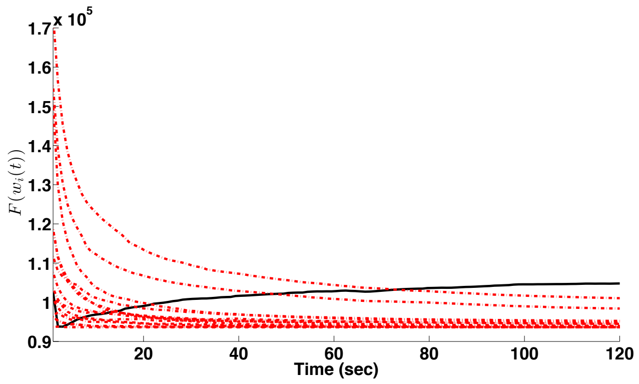

Fig. 2. Network of 15 nodes solving problem (14) on a graph where node 1 has many more neighbours that the rest of the nodes. Because of asynchronism and delays, the resulting updates do not correspond to a doubly stochastic matrix, and the nodes do not reach consensus on the average. Consequently, node 1 (solid black line) does not converge to right solution and the algorithm does not solve the problem.

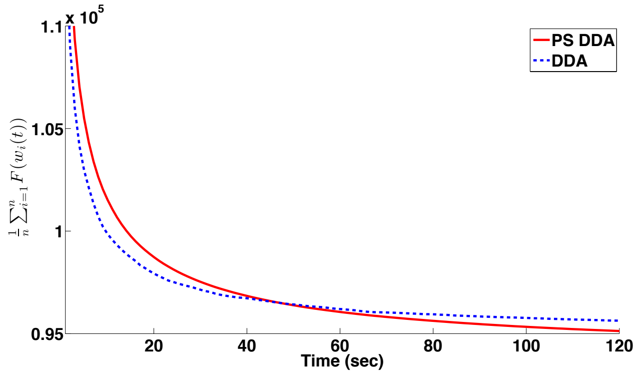

Fig. 3. (Blue dashed) The average performance of the team produced by running DDA and ignoring the weights (average of plots inFigure 2). (Red Line) When the weights w are not kept to 1 and true asynchronous PS-DDA is running, the network converges to the right solution despite the asymmetry of communication overhead of its nodes.

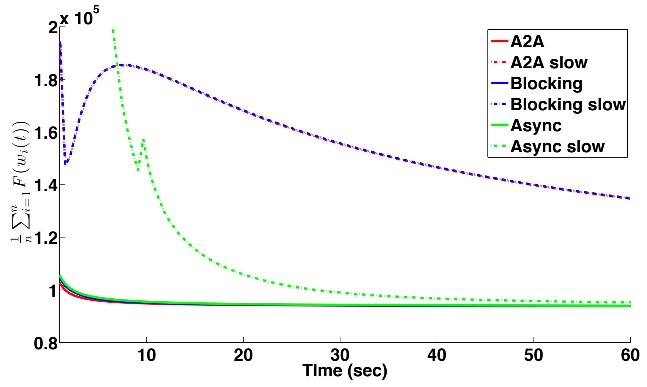

Fig. 4. (Red,Blue,Green solid) Average progress towards the solution of problem (14) with All2All, PS-DDA blocking and PS-DDA asynchronous algorithms when all nodes are running at full speed. (Red,Blue,Green dashed) Average progress when one node is artificially slowed down to have a 0.5 second delay at each local iteration. In this case the synchronous algorithms are running at the speed of the slow node and their curves are overlapping. The asynchronous algorithms is significantly less affected by the slow node.

one function call. However, the call is blocking. Hence, using All2All implicitly requires a synchronous approach and all nodes must call All2All simultaneously. This solution resembles algorithms baed on the popular Map-Reduce approach that have been developed recently for distributed optimization [32], [33]. We compare this solution with our PS-DDA implementation operating in blocking and asynchronous mode, executing over a communication G corresponding to the complete graph. In blocking mode, at each iteration each node blocks in a receive call until one message from each neighbour arrives. This turns PS-DDA into a synchronous algorithm and each node computes the exact average $z(t)$ at every iteration.

Figure 4 shows two sets of experiments. Initially, we solve problem (14) with all three algorithms in a delay-free environment. As we see from the solid lines at the bottom of the figure, the All2All implementation is slightly faster than the blocking PS-DDA which again is just slightly faster than the asynchronous PS-DDA implementation. Then, to illustrate the benefits of asynchronism, we artificially slow down one of the nodes by adding a 0.5 second pause after each local gradient computation. In practice, a node could be slowed down because it is spending cycles on unrelated tasks, it could have more data to process, or it could simply have a less powerful CPU. It should come to no surprise that the synchronous algorithms (purple and red dashed lines) are both severely impacted as their overall computation runs at the speed of the slowest node. However, the asynchronous algorithm (green dashed line) is not as affected. The fast nodes quickly converge to the solution and the slow node is pulled to the right solution.

VII. Concluding Remarks and Future Work

Consensus-based distributed optimization algorithms are an attractive alternative for solving large-scale problems over peer-to-peer networks. The advantages of such an approach are increased robustness to node failures and scalability. The main difficulty comes from the lack of sophisticated communication infrastructure, and performance heavily depends on the network properties.

We identify averaging, one directional communication, and asynchronism as the three key ingredients that a consensus algorithm should contain for a reliable and efficient practical implementation. We note that the benefit of an algorithm that accommodates real network constraints comes at the price of more difficult mathematical analysis. In the second part of the paper, we discuss additional implementation issues we have observed/experienced, including numerical instabilities and de-synchronizing the step sizes of different nodes. For these issues we explain the root cause and side effects as well as ways to prevent them from happening. In the last part, we present two simulations on a real cluster that illustrate how the key ingredients mentioned above yield practical algorithms that work correctly even in adverse circumstances where the communication overhead of the nodes is not equally distributed or where nodes experience different and unbalanced workloads.

In the future, we plan to conduct a more thorough theoretical investigation of the effect of de-synchronizing step sizes, as well as thresholding the Push Sum weights to avoid numerical instabilities. We also intend to scale our experiments to a significantly larger cluster with many more than the modes 15 nodes used in the experiments reported here.

References

[1] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling up Machine Learning, Parallel and Distributed Approaches*. Cambridge University Press, 2011.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010.

[3] J. Duchi, A. Agarwal, and M. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic Control*, vol. 57, no. 3, pp. 592–606, 2011.

[4] S. S. Ram, A. Nedic, 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, 2011.

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

[6] B. Johansson, M. Rabi, and M. Johansson, "A randomized incremental subgradient method for distributed optimization in networked systems," *SIAM Journal on Control and Optimization*, vol. 20, no. 3, 2009.

[7] J. Chen and A. Sayed, "Diffusion adaptation strategies for distributed optimization and learning over networks," *IEEE Transactions on Signal Processing*, vol. 60, no. 8, pp. 4289–4305, August 2012.

[8] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847 – 1864, November 2010.

[9] J. N. Tsitsiklis, D. P. 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.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[11] D. Jakovetic, J. Xavier, and J. M. Moura, "Fast distributed gradient methods," *arXiv:1112.2972v1*, 2011.

[12] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *51st IEEE Conference on Decision and Control*, 2012.

[13] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," in *Proceedings of the IEEE*, vol. 95:1, 2007, pp. 215 – 233.

[14] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis, "Convergence in multiagent coordination, consensus, and flocking," in *IEEE Conference on Decision and Control*, 2006, pp. 2996 – 3000.

[15] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, pp. 2508–2530, 2006.

[16] T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione, "Broadcast gossip algorithms for consensus," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2748 – 2761, July 2009.

[17] A. Tahbaz-Salehi and A. Jadbabaie, "Necessary and sufficient conditions for consensus over random independent and identically distributed switching graphs," in *Proceedings of the 46th IEEE Conference on Decision and Control*, 2007.

[18] V. M. Preciado, A. Tahbaz-Salehi, and A. Jadbabaie, "On asymptotic consensus value in directed random networks," in *49th IEEE Conference on Decision and Control*, Atlanta, GA, USA, December 2010.

[19] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *FOCS, vol. 44. IEEE Computer Society Press, pp. 482–491*, 2003.

[20] F. Benezit, V. Blondel, P. Thiran, J. Tsitsiklis, and M. Vetterli, "Weighted gossip: Distributed averaging using non-doubly stochastic matrices," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2010, pp. 1753 – 1757.

[21] K. I. Tsianos and M. G. Rabbat, "Distributed dual averaging for convex optimization under communication delays," in *American Control Conference (ACC)*, 2012.

[22] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Journal on Control and Optimization*, vol. 48, No 1, pp. 33–55, 2009.

[23] K. I. Tsianos and M. G. Rabbat, "The impact of communication delays on distributed consensus algorithms," *Submitted to Transactions on Automatic Control (http://arxiv.org/abs/1207.5839)*, 2012.

[24] ——, "Distributed consensus and optimization under communication delays," in *49th Allerton Conference on Communication, Control, and Computing*, 2011.

[25] A. Nedic and A. Ozdaglar, "Convergence rate for consensus with delays," *Journal of Global Optimization*, vol. 47, no. 3, pp. 437–456, 2010.

[26] B. Gharesifard and J. Cortes, "When does a digraph admit a doubly stochastic adjacency matrix?" in *Proceedings of the American Control Conference*, Baltimore, Maryland, 2010, pp. 2440–2445.

[27] E. Seneta, *Non-negative Matrices and Markov Chains*. Springer, 1973.

[28] J. A. Fill, "Eigenvalue bounds on convergence to stationarity for non reversible markov chains, with an application to the exclusion process," *The Annals of Applied Probability*, vol. 1, no. 1, pp. 62–87, 1991.

[29] R. Montenegro and P. Tetali, *Mathematical Aspects of Mixing Times in Markov Chains*. Foundations and Trends in Theoretical Computer Science (Vol 1, No 3), 2006.

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

[31] B. Touri, "Product of random stochastic matrices and distributed averaging," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2011.

[32] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM Magazine - 50th anniversary issue*, vol. 51, no. 1, pp. 107–113, 2008.

[33] A. Agarwal, O. Chapelle, M. Dudik, and J. Langford, "A reliable effective terascale linear learning system," Feb 2012, submitted, available online as arxiv:1110.4198.