Distributed Optimization Algorithm Adapting to Latency in Network Links

Kush Prasad, Advisor: Dr. Yufeng Xin

***Abstract*— Distributed optimization algorithms for a network of distributed agents have proved to be important in the areas of machine learning, sensor networks and big data among other applications. In this paper, we present a review of the alternating direction method of multipliers (ADMM), a method which has been gaining increasing attention recently to solve distributed optimization problems. We present the optimization problem formulation under study, the assumptions made, the ADMM algorithm and its convergence properties. We focus on performance advantages in solving common optimization problems (global consensus variable optimization and the sharing problem) and highlight the application of ADMM to particular problems in linear classification and wireless sensor networks. We also describe the asynchronous and decentralized variants of ADMM. In practical applications in distributed networked systems, the standard ADMM algorithm may not be ideal for application since the distributed agents are assumed to be synchronous in the standard ADMM. We finally formulate a distributed optimization problem, and use the ADMM algorithm to give its solution.**

I. INTRODUCTION

Multi-agent optimization problems find application in various kinds of networks such as wireless sensor networks [11], power grids [8], cluster of machines in a datacenter [13] and chips on motherboard of a machine. In such networks, while achieving a global optimization goal, distributed optimization helps save communication bandwith and energy which is a scarce resource in networks. In the special case of data-based applications, with the availability of large-scale data from the internet, one machine is not able to store all the data and the data is spread across multiple (maybe upto a thousand) machines. Since transferring such high amounts of data would incur high communication costs and may not even be feasible, it is advisable to perform optimization at the local machines using distributed optimization algorithms [1]. More specifically, we will focus on solving the optimization problem where the objective function consists of sum of separable convex functions having its own set of constraints. The problem is formulated as below.

$$\min_{x_1,..,x_N} \sum_{i=1}^{N} f_i(x_i) \tag{1}$$

Subject to:

$$x \in X \tag{2}$$

In the equation above, $x^T = [x_1^T, .., x_N^T]$ and lies in the constraint set X. x, x_1, x_N here are column vectors. From an application perspective, each function f_i resides on a separate node in a network. A variety of problems such as the consensus optimization problem in a network can be formulated in the above framework. One class of methods for solving such problems are sub-gradient methods [10], [5], [9] in which a node uses the sub-gradient of its objective function and the latest iterates from its neghbours to compute its iterate. The best known rate of convergence for these methods is of the order of $O\left(\frac{1}{\sqrt{k}}\right)$ [6] where k is number of iterations. Another class of methods involves solving primal and dual problems alternately. A prominent algorithm of this class is the alternate direction method of multipliers (ADMM). The best known convergence rate for ADMM is of the order of $O\left(\frac{1}{k}\right)$ [2], [4], [12]. The algorithm finds application in a variety of areas such as image processing, machine learning, processing of large-scale data, wide area networks among other applications. It is specifically suitable for applications which require an optimization problem to be solved in a decentralized manner across different machines.

In this work we study a decentralized version of the ADMM algorithm where global optimization is achieved by communicating with only the neighboring nodes. This version of the algorithm [11], uses only a subset of nodes (called bridge nodes) to communicate with all its neighbors while nodes which are not bridge nodes communicate only with the bridge nodes. This saves communication costs in the network. We consider the problem of latency and failure in network links and develop a dynamic algorithm which adapts to latency in links in real-time. Our algorithm allows a certain bridge node in the network to communicate only with a subset of its neighbors. We show that the algorithm still converges to an optimal solution retaining the original convergence properties. Through the experiments we also show that our algorithm results in lower running time for computing the optimal solution.

In section II of this paper, we present the ADMM algorithm along with its assumptions and convergence properties. In section III, we describe a decentralized ADMM algorithm in which nodes only communicate with the neighboring nodes. In Section IV, we discuss a policy and an algorithm to account for for latency in network links and link failures. In section V, we present our conclusions regarding the algorithms presented and the inference from our results.

II. Background on ADMM Formulation, Algorithm and Application

A. ADMM Problem Formulation

In this section, we describe the ADMM problem formulation and the iterative algorithm used to solve it. We specifically use ADMM when the objective function is separable into different functions and each of the function itself is convex. Each separable function being convex ensures that we can compute its minimum separately. The Standard ADMM is formulated as below.

$$\min_{x_1,\ldots,x_N,z} \left(\sum_{i=1}^{N} f(x_i)\right) + g(z) \tag{3}$$

Subject to:

$$A_i x_i + B_i z = c_i \;\; \forall\, i = 1, \ldots, N \tag{4}$$

In the above equation, $x_i \in \mathbb{R}^n_i$, $z \in \mathbb{R}^m$, $A_i \in \mathbb{R}^{p_i \times n_i}$, $B_i \in \mathbb{R}^{p_i \times m}$, $c_i \in \mathbb{R}^{p_i}$. As can be seen above, the objective function has been split into separate functions $f_i(x_i)$ and $g(z)$ and the optimization variable has been split into sub-vectors of the original variable. This separation of objective function and variables helps us solve the problem in a distributed manner.

B. Algorithm

To solve any optimization problem using the ADMM methodology, we form its augmented lagrangian. The augmented lagrangian L_ρ for the problem formulation in equations 3 - 4 is given by equation 5. Minimizing the augmented lagrangian is equivalent to minimizing the original optimization problem. We minimize the augmented lagrangian by first optimizing it with respect to primal variables by solving a convex optimization problem and then with respect to the dual variable using gradient ascent. The iterations to update the variables are given by equations 6 - 8. We shall mention here that the constraints given by equation 4 may also be coupled in the variables x_i. The updates for x_i can then be solved keeping x_{-i} constant and optimizing over x_i where $x_{-i} = [x_1, .., x_{i-1}, x_{i+1}, \ldots, x_N]$. In the following equations, $\rho > 0$ is the penalty constant and each x_i and z updates are separate convex optimization problems in themselves.

$$\begin{aligned} L_\rho(x_1, \ldots, x_N, z, y_1, \ldots, y_N) &= \sum_{i=1}^{N} f(x_i) \\ &+ g(z) + \sum_{i=1}^{N} y_i^T (A_i x_i + B_i z - c_i) \\ &+ \sum_{i=1}^{N} (\rho/2) \|A_i x_i + B_i z - c_i\|_2^2 \end{aligned} \tag{5}$$

$$x_{i(k+1)} := arg\min_{x_i}\ L_\rho(x_i, z_k, y_{i(k)})\ \forall\, i \quad (6)$$

$$\begin{aligned} z_{k+1} := arg\min_{z}\ L_\rho(&x_{1(k+1)}, .., x_{N(k+1)} \\ &z, y_{1(k)}, .., y_{N(k)}) \end{aligned} \quad (7)$$

$$\begin{aligned} y_{i(k+1)} := y_{i(k)} + \rho(A_i x_{i(k+1)} + B_i z_{i(k+1)} - c_i) \\ \forall\, i \end{aligned} \quad (8)$$

C. Optimality Conditions, Convergence and Stopping Criterion

The ADMM algorithm works and converges under the assumptions that the separable functions are closed, proper and convex and that there exists a saddle point for the unaugmented lagrangian. When these assumptions mentioned are met, the ADMM iterations (equations 6 - 8) ensure convergence of objective function to its optimum value and the value of dual variable to its optimal value when the number of iterations tends to ∞. In practice the ADMM is slow to converge to an optimal value within a small error. However, the algorithm converges to modest accuracy in tens of iterations which is what is required in applications such as estimation and machine learning. The optimality conditions for the ADMM problem formulation are primal feasibility and dual feasibility. They can be stated as below. The reader shall refer to work by Stephen Boyd et al [2] for details of the derivation although they can also be from the optimization problem formulation in equations 3 - 4 and using equations 5 - 8.

Optimality Conditions:

1) $A_i x_i + B_i z \to c_i\ \ \forall\, i = 1, ..., N$
2) $\rho A_i^T B_i(z_{k+1} - z_k) \to 0\ \ \forall\, i = 1, ..., N$

D. Implementation in a Network

ADMM finds application in various areas of machine learning [14] [3], estimation and sensor networks since it enables us to solve the optimization problem in a distributed manner. Solving a problem in a distributed manner is particularly helpful when solving machine learning problems where data is spread across multiple machines and also for solving problems of estimation in sensor networks. The optimization problem may be solved using a star-like topology where a master node is connected to and communicates with several worker nodes. The pseudocodes for the operations at the master and worker nodes is given by algorithms **??** and 2 respectively. The function of the master node is to receive updates of $x_{i(k+1)}$, $y_{i(k)}$ from each worker node, use their values to compute z_{k+1} and then send z_{k+1} to each worker node. Each worker node computes $y_{i(k)}$, $x_{i(k+1)}$ using the received value of z_k and sends $y_{i(k)}$, $x_{i(k+1)}$ to the master node.

Algorithm 1 Algorithm at Worker Node i

```
Initialize $x_{i(k+1)}$ and $y_{i(k)}$
flag = 1
while flag = 1 do                                              ▷
flag = 0 when optimality conditions are met. It
is received from master node in each iteration.
    Receive $z_k$, value of flag from master node.
    Compute $y_{i(k)}$.
    Compute $x_{i(k+1)}$.
    Send $x_{i(k+1)}$, $y_{i(k)}$ to master node.
end while
```

Algorithm 2 Algorithm at Master Node i

```
Initialize $x_{i(k+1)}$ and $y_{i(k)}$.
while Optimality$(x_1, .., x_N, z, y_1, .., y_N)$ = 0 do
    Compute $z_k$ using $x_{i(k+1)}$ and $y_{i(k)}$.
    Send $z_k$ to each worker node.
    Receive $x_{i(k+1)}$ and $y_{i(k)}$.
end while
NOTE: Optimality$(x_1, .., x_N, z, y_1, .., y_N)$ is a
function which checks the optimality conditions
and returns 1 when satisfied otherwise 0.
```

III. ADMM using Bridge Nodes in a Network

In sensor networks communicating data to a central node costs energy which is a precious resource in such networks. Decentralized optimization algorithms depend only on communication among the neighboring nodes and thus consume less energy in comparison to centralized algorithms. Application areas which use these algorithms are machine learning and parameter estimation in sensor networks. There has been research on decentralized ADMM algorithms [12], [7] in the recent past.

In this section we are interested in consensus in ad hoc wireless sensor networks and the details presented are based on the research in [11]. We present how a network parameter can be estimated using observations collected across nodes in the network. To be specific, we have J nodes in the network. We denote the neighbors of node j as N_j. We know that the node links are symmetric, i.e. if node i can communicate with node j, then node j can communicate with i ($i \in N_j$ if and only if $j \in N_i$). Our objective is to estimate a $\mathbb{R}^{p\times 1}$ vector s based on observations $\{x_j \in \mathbb{R}^{L_j\times 1}\}_{j=1}^{J}$ taken at node j and following the probability distribution function $p_j(x_j; s)$. The maximum likelihood estimator is then give by the following.

$$\hat{s}_{ML} = arg \min_{s\in\mathbb{R}^{p\times 1}} - \sum_{j=1}^{J} \ln[p_j(x_j; s)] \tag{9}$$

The above model is applicable when the pdf's for observations at each node j are known or have been assumed. We will present decentralized iterative algorithms to solve the above optimization problem using only single-hop communication. We make the following two assumptions which will help us in solving the problem in a decentralized manner.

Assumptions:

- The communication graph of nodes is connected. There is a path connecting any two nodes.
- The pdf's $p_j(x_j; s)$ are log-concave with respect to the network parameter s.

Assumption 1 ensures network connectivity which in turn ensures that all observations are used to compute the optimal solution. And assumption 2 ensures that there is a unique optimal solution to the global problem and the global problem is strictly convex. Let us now formulate the problem in a decentralized manner. Since the summands in equation 9 are coupled through the parameter vector s for each node j, we introduce an auxiliary variable s_j and formulate the problem in the following manner.

$$\{\hat{s}_j\}_{j=1}^{J} = arg \min_{s_j} - \sum_{j=1}^{J} \ln[p_j(x_j; s)] \tag{10}$$

Subject to:

$$s_j = \bar{s}_b, \quad b \in B, \quad j \in N_b \tag{11}$$

Where $B \subset [1, J]$ is a subset of bridge nodes maintaining local vectors $\hat{s}_b$. These vectors help to maintain consensus among the neighboring nodes. Let us now give the definition of B such that it is a subset of bridge nodes. Set B is a subset of bridge nodes if and only if:

1) Every node has a neighboring node which is a bridge node
2) Any pair of single-hop neighboring nodes must share a bridge node.

The above definition for B provides a necessary and sufficient condition for the equivalence of optimization formulation in 9 and the formulation in 10. The lagrangian for equation 10 can be written in the following way.

$$\begin{aligned} L(s, \bar{s}, v) = & - \sum_{j=1}^{J} \ln[p_j(x_j; s)] \\ & + \sum_{b\in B} \sum_{j\in N_b} (v_j^b)^T (s_j - \bar{s}_b) \\ & + \sum_{b\in B} \sum_{j\in N_b} \frac{c_j}{2} \|s_j - \bar{s}_b\|_2^2 \end{aligned} \tag{12}$$

Where $s = \{s_j\}_{j=1}^{J}$, $\bar{s} = \{\bar{s}_b\}_{b\in B}$, $v = \{v_j^b\}_{j\in[1,J]}^{b\in B_j}$ and $\{c_j\}_{j=1}^{J}$ are constants greater than zero. Applying alternate direction method of multipliers to equation 12, we get the following iterations for s_j, $\bar{s}_b$ and v_j^b.

$$\begin{aligned} s_j(k+1) = arg\min_{s_j} & - \ln p_j(x_j; s) \\ & + \sum_{b \in B_j} (v_j^b)^T (s_j - \bar{s}_b(k)) \\ & + \sum_{b \in B_j} \frac{c_j}{2} \|s_j - \bar{s}_b(k)\|_2^2 \end{aligned} \tag{13}$$

$$\bar{s}_b(k+1) = \sum_{j \in N_b} \frac{1}{\sum_{\beta \in N_b} c_\beta} [v_j^b(k) + c_j s_j(k+1)] \quad \forall b \in B \tag{14}$$

$$\begin{aligned} v_j^b(k+1) = v_j^b(k) + c_j(s_j(k+1) \\ - \bar{s}_b(k+1)), \quad \forall b \in B_j \end{aligned} \tag{15}$$

The parameter estimates s_j and $\{v_j^b\}_{b \in B_j}$ are maintained at the j^{th} node and consensus estimates $\bar{s}_b$ are maintained at the corresponding bridge nodes. The bridge node communicates $\bar{s}_b$ to the neighboring nodes N_b and the nodes j communicate s_j to bridge nodes $b \in B_j$. The above iterations are derived from the ADMM algorithm. The parameter estimates s_j at nodes and the estimates $\bar{s}_b$ at the bridge nodes converge to a consensus value as the number of iterations tend to ∞. After enough iterations each node has the parameter estimate s_j which is a solution to equation 9. Since the pdf's at each node are concave the solution is unique. Thus we have the following.

$$\lim_{k \to \infty} s_j(k) = \lim_{k \to \infty} \bar{s}_b(k) = \hat{s}_{ML} \ \ \forall j \in [1, J], \ \ b \in B \tag{16}$$

We shall note that only single-hop communications take place to run the ADMM iterations. Thus the sensor network consumes less power and and is robust to failures in comparison to a system where there is a central processor. For practical implementation, $x_i(k+1)$ and dual variables $y_i^b(k)$ are updated at all nodes i in the network. These updates are communicated to bridge nodes $b \in NB(i)$ in the network and $x_b(k+1)$ is then updated at the bridge nodes. The bridge node communicates back its update $x_b(k+1)$ to all nodes $i \in B(b)$. Thus we see that we can implement the ADMM algorithm using bridge nodes in networks. This releases the communication load from one central node. In certain networks, there may be some machines which have more resources than others (in terms of memory, computation and bandwidth), they can be allotted as the bridge nodes in the network.

IV. Accounting for Latency and Failures in Network Links

All real networks have latency in their links. It is a well known fact that the fastest speed of communication (theoretically) is the speed of light and that the actual time taken to communicate is actually significantly more in practice. Latency in links, failure of links and failure of nodes in the network affect the performance of any distributed algorithm running on a network. ADMM with bridge nodes described in the previous section, when applied on a real network will be affected by issues of latency in links, failure of links and node failures. The failures may be more prominent in certain kinds of networks such as long-distance networks and wireless networks. To overcome the issues of failure of nodes and latency in links, we are interested in developing a dynamic algorithm which adapts in real-time to failures in the network and latency in links. We note that different links have different latency values at different times in a network. Most probably all links are not going to fail at the same time or the latency in all links in the network is not going to be high at the same time. This observation can be used to develop a distributed optimization algorithm which adapts to node and link failures and latency issues in links in real-time.

A. Policy and Algorithm to Adapt to Latency in Links

In the alternate direction method of multipliers with bridge nodes, computation is run with all bridge nodes acting as the master nodes performing computation and all respective neighboring nodes as the worker nodes. Thus, if we use the algorithm to build consensus in a network, all the bridge nodes perform computation to build consensus among its neighbors. A bridge node communicates with all its neighbors while all the non-bridge nodes communicate only with their bridge nodes. We are suggesting a policy that a node in the network may apply to adapt to latency in its link with the bridge node. When a node detects high latency value in its link with the bridge node, it shall either ask one if its neighbors (in common with the bridge node) to become the bridge node or itself become the bridge node. And after finding a new bridge node, the node does not communicate the bridge node with which it has latency issues. Thus we modify the algorithm described in the previous section such that a bridge node does not communicate with all its neighboring nodes but only a subset of its neighboring nodes. This is a key point in the development of our algorithm and shall be emphasized. The policy and the algorithm shall satisfy certain properties of graph connectivity and bridge nodes which were satisfied by algorithm in the previous section. Provided certain conditions, this will enable communication of the nodes using links which have lower latency.

Below is the notation we use to describe our policy. Algorithm 3 gives the specific pseudocode to describe our algorithm.

1) $Bridge(i) = 1, 0$ indicates whether the node i is a bridge node or not.
2) $Lmax(i)$ = maximum latency among all links of node i.
3) $Lmax_{no-bridge}(i)$ = maximum latency among all links (but the link with bridge node).
4) $minLmaxNeigh(i)$ = minimum of all Lmax values received from neighboring nodes.
5) If $latency_{bridge}(b)$ = latency in link between the node and its bridge node b.
6) $latency_{limit}$ = maximum allowable latency value in link with bridge node.
7) $N(i)$ = set of neighboring nodes of node i.
8) $N_B(b, i)$ = set of node in $N(i)$ which also have bride node b.
9) $B(i)$ = set of bridge nodes of node i.

Algorithm 3 Algorithm at node i

```
for all $b \in B(i)$ do
    if $latency_{bridge}(b) > latency_{limit}$ & $N_B(b, i) \neq \emptyset$ then
        if $Bridge(i)$ = 1 then
            remove $b$ from $B(i)$.
        else $Bridge(i)$ = 0
            $minLmaxNeigh(i) = \min_{n} Lmax(n \in N_B(b, i))$
            if $Lmax_{no-bridge}(i) < minLmaxNeigh(i)$ then
                $Bridge(i) = 1$
                inform all $n \in N(i)$ that node $i$ is now a bridge node.
                remove $b$ from $B(i)$.
            else $Lmax_{no-bridge}(i) \geq minLmaxNeigh(i)$
                for all $j \in N_B(b, i)$ do
                    if $Lmax_{no-bridge}(i) \geq Lmax(j)$ & $Lmax(j) < latency_{bridge}(b)$ & $Lmax(j) < latency_{limit}$ then
                        remove node $b$ from $B(i)$.
                        add node $j$ to $B(i)$.
                    end if
                end for
            end if
        end if
    end if
end for
```

B. Practical Implementation Details

We discuss two practical implementation details for implementation of the bridge node algorithm which adapts to adapts to latency in links in the network. One of them is an algorithm for initializing the bridge nodes and the other is an algorithm for computing latency in all the links of a node. The network needs to initialize bridge nodes in the network using a distributed fashion while also satisfying the properties of graph connectivity and bridge node properties. We describe such a process in algorithm 4 for initializing the bridge nodes.

Implementation of algorithm 3 requires computation of latency in all links of the node i. We propose

Algorithm 4 Algorithm for Initialization at node i

```
b = 0
for $n \in N(i)$ do
    b = Bridge(n)
    if b = 1 then
        Bridge(i) = 0
        End for loop
    end if
end for
if b = 0 then
    Bridge(i) = 1
else
    Bridge(i) = 0
end if
```

the following procedure to do that.

1) Run threads on node i that can simultaneously communicate with the neighboring nodes $N(i)$.
2) Send messages to neighboring nodes and record the time in each thread.
3) Receive messages from neighboring nodes and record the time as well.
4) The difference in the two values of the time recorded gives us an estimate of the latency value.

V. Conclusions

In this work, we described the described the distributed optimization problem and how the alternate distributed method of multipliers (ADMM) can be applied to solve the distributed optimization problem. ADMM with a single master node is susceptible to failures and we thus show a more robust application of the algorithm where only the neighboring nodes communicate with each other. A set of nodes in the network function as the master node with their respective neighbors as worker nodes. This reduces the communication cost in the network while making the algorithm robust to node failures. We developed a dynamic algorithm which adapts to latency and failure in network links. We point that a bridge node may communicate with only a subset of its nodes while maintaining graph connectivity. The algorithm functions in such a way that the convergency properties of the algorithm are maintained. The algorithm has a lower runtime in comparison to the original algorithm as shown by the experiments. We thus come to the conclusion that, in the presence of excess latency in network links, it is better (in terms of runtime performance) for a bridge node to communicate with only a subset of its neighbors. This insight results in a dynamic algorithm and policy which we have described in this work.

References

[1] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., 1989.

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[3] Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 289–297, 2013.

[4] Bingsheng He and Xiaoming Yuan. On the o(1/n) convergence rate of the douglas-rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[5] Björn Johansson, Tamás Keviczky, Mikael Johansson, and Karl Henrik Johansson. Subgradient methods and consensus algorithms for solving convex optimization problems. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4185–4190. IEEE, 2008.

[6] Ilan Lobel and Asuman Ozdaglar. Convergence analysis of distributed subgradient methods over random networks. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 353–360. IEEE, 2008.

[7] João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Puschel. D-admm: A communication-efficient distributed algorithm for separable optimization. *Signal Processing, IEEE Transactions on*, 61(10):2718–2723, 2013.

[8] Seyedbehzad Nabavi, Jianhua Zhang, and Aranya Chakrabortty. Distributed optimization algorithms for wide-area oscillation monitoring in power systems using interregional pmu-pdc architectures.

[9] Angelia Nedić and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009.

[10] S Sundhar Ram, A Nedić, and Venugopal V Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147(3):516–545, 2010.

[11] Ioannis D Schizas, Alejandro Ribeiro, and Georgios B Giannakis. Consensus in ad hoc wsns with noisy linkspart i: Distributed estimation of deterministic signals. *Signal Processing, IEEE Transactions on*, 56(1):350–364, 2008.

[12] Ermin Wei and Asuman Ozdaglar. Distributed alternating direction method of multipliers. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 5445–5450. IEEE, 2012.

[13] Hong Xu and Baochun Li. Joint request mapping and response routing for geo-distributed cloud services. In *INFOCOM, 2013 Proceedings IEEE*, pages 854–862. IEEE, 2013.

[14] Caoxie Zhang, Honglak Lee, and Kang G Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *International Conference on Artificial Intelligence and Statistics*, pages 1398–1406, 2012.