Distributed Solution of Large-Scale Linear Systems via Accelerated Projection-Based Consensus

Navid Azizan-Ruhi, *Member, IEEE*, Farshad Lahouti, *Senior Member, IEEE*, Amir Salman Avestimehr, *Senior Member, IEEE*, and Babak Hassibi, *Member, IEEE*

***Abstract*—Solving a large-scale system of linear equations is a key step at the heart of many algorithms in scientific computing, machine learning, and beyond. When the problem dimension is large, computational and/or memory constraints make it desirable, or even necessary, to perform the task in a distributed fashion. In this paper, we consider a common scenario in which a taskmaster intends to solve a large-scale system of linear equations by distributing subsets of the equations among a number of computing machines/cores. We propose a new algorithm called *Accelerated Projection-based Consensus*, in which at each iteration every machine updates its solution by adding a scaled version of the projection of an error signal onto the nullspace of its system of equations, and the taskmaster conducts an averaging over the solutions with momentum. The convergence behavior of the proposed algorithm is analyzed in detail and analytically shown to compare favorably with the convergence rate of alternative distributed methods, namely distributed gradient descent, distributed versions of Nesterov's accelerated gradient descent and heavy-ball method, the block Cimmino method, and Alternating Direction Method of Multipliers. On randomly chosen linear systems, as well as on real-world data sets, the proposed method offers significant speed-up relative to all the aforementioned methods. Finally, our analysis suggests a novel variation of the distributed heavy-ball method, which employs a particular distributed preconditioning and achieves the same theoretical convergence rate as that in the proposed consensus-based method.**

***Index Terms*—System of linear equations, distributed computing, big data, consensus, optimization.**

Manuscript received June 25, 2018; revised December 14, 2018 and April 2, 2019; accepted April 26, 2019. Date of publication June 4, 2019; date of current version June 21, 2019. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Laura Cottatellucci. This work was supported in part by the National Science Foundation under Grants CCF-1423663, CCF-1409204, and ECCS-1509977, in part by a grant from Qualcomm Inc., in part by NASA's Jet Propulsion Laboratory through the President and Director's Fund, and in part by fellowships from Amazon Web Services, Inc., and PIMCO, LLC. This paper was presented in part at the IEEE International Conference on Acoustics, Speech, and Signal Processing, Calgary, AB, Canada, April 2018 [1]. *(Corresponding author: Navid Azizan-Ruhi.)*

N. Azizan-Ruhi is with the Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: azizan@caltech.edu).

F. Lahouti and B. Hassibi are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: lahouti@caltech.edu; hassibi@caltech.edu).

A. S. Avestimehr is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90007 USA (e-mail: avestimehr @ee.usc.edu).

Digital Object Identifier 10.1109/TSP.2019.2917855

I. Introduction

With the advent of big data, many analytical tasks of interest rely on distributed computations over multiple processing cores or machines. This is either due to the inherent complexity of the problem, in terms of computation and/or memory, or due to the nature of the data sets themselves that may already be dispersed across machines. Most algorithms in the literature have been designed to run in a sequential fashion, as a result of which in many cases their distributed counterparts have yet to be devised. In order to devise efficient distributed algorithms, one has to address a number of key questions such as (a) What computation should each worker carry out, (b) What is the communication architecture and what messages should be communicated between the processors, (c) How does the distributed implementation fare in terms of computational complexity, and (d) What is the rate of convergence in the case of iterative algorithms.

In this paper, we focus on solving a large-scale system of linear equations, which is one of the most fundamental problems in numerical computation, and lies at the heart of many algorithms in engineering and the sciences. In particular, we consider the setting in which a taskmaster intends to solve a large-scale system of equations in a distributed way with the help of a set of computing machines/cores (Figure 1). This is a common setting in many computing applications, and the task is mainly distributed because of high computational and/or memory requirements (rather than physical location as in sensor networks).

This problem can in general be cast as an optimization problem, with a cost function that is separable in the data[1] (but not in the variables). Hence, there are general approaches to construct distributed algorithms for this problem, such as distributed versions of gradient descent [2]–[4] and its variants (e.g. Nesterov's accelerated gradient [5] and heavy-ball method [6]), as well as the so-called Alternating Direction Method of Multipliers (ADMM) [7] and its variants. ADMM has been widely used [8]–[10] for solving various convex optimization problems in a distributed way, and in particular for consensus optimization [11]–[13], which is the relevant one for the type of separation that we have here. In addition to the optimization-based methods, there are a few distributed algorithms designed specifically

[1]Solving a system of linear equations, $Ax = b$, can be set up as the optimization problem $\min_x \|Ax - b\|^2 = \min_x \sum_i \|(Ax)_i - b_i\|^2$.

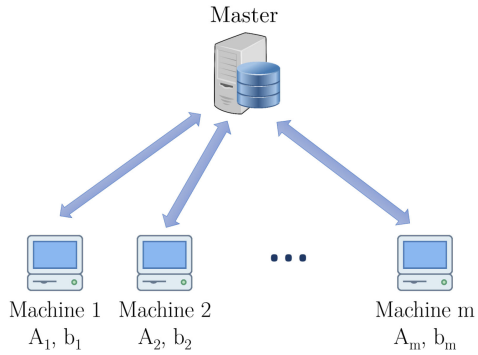

Fig. 1. Schematic representation of the taskmaster and the m machines/cores. Each machine i has only a subset of the equations, i.e. $[A_i, b_i]$.

for solving systems of linear equations. The most famous one of these is what is known as the block Cimmino method [14]–[16], which is a block row-projection method [17], and is in a way a distributed implementation of the Kaczmarz method [18]. Another algorithm has been recently proposed in [19], [20], where a consensus-based scheme is used to solve a system of linear equations over a network of autonomous agents. Our algorithm bears some resemblance to all of these methods, but as it will be explained in detail, it has much faster convergence than any of them.

Our main contribution is the design and analysis of a new algorithm for distributed solution of large-scale systems of linear equations, which is significantly faster than all the existing methods. In our methodology, the taskmaster assigns a subset of equations to each of the machines and invokes a distributed consensus-based algorithm to obtain the solution to the original problem in an iterative manner. At each iteration, each machine updates its solution by adding a scaled version of the projection of an error signal onto the nullspace of its system of equations, and the taskmaster conducts an averaging over the solutions with momentum. The incorporation of a momentum term in both projection and averaging steps results in accelerated convergence of our method, compared to the other projection-based methods. For this reason, we refer to this method as *Accelerated Projection-based Consensus (APC)*. We provide a complete analysis of the convergence rate of APC (Section III), as well as a detailed comparison with all the other distributed methods mentioned above (Section IV). Also by empirical evaluations over both randomly chosen linear systems and real-world data sets, we demonstrate the significant speed-ups from the proposed algorithm, relative to the other distributed methods (Section VI). Finally, as a further implication of our results, we propose a novel distributed preconditioning method (Section VII), which can be used to improve the convergence rate of distributed gradient-based methods.

II. The Setup

We consider the problem of solving a large-scale system of linear equations

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{N\times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^N$. While we will generally take $N \geq n$, we will assume that the system has a unique solution. For this reason, we will most often consider the square case ($N = n$). The case where $N < n$ and there are multiple (infinitely many) solutions is discussed in Section V.

As mentioned before, for large-scale problems (when $N, n \gg 1$), it is highly desirable, or even necessary, to solve the problem in a distributed fashion. Assuming we have m machines (as in Figure 1), the equations can be partitioned so that each machine gets a disjoint subset of them. In other words, we can write (1) as

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix},$$

where each machine i receives $[A_i, b_i]$. In some applications, the data may already be stored on different machines in such a fashion. For the sake of simplicity, we assume that m divides N, and that the equations are distributed evenly among the machines, so that each machine gets $p = \frac{N}{m}$ equations. Therefore $A_i \in \mathbb{R}^{p\times n}$ and $b_i \in \mathbb{R}^p$ for every $i = 1, \dots m$. It is helpful to think of p as being relatively small compared to n. In fact, each machine has a system of equations which is highly under-determined.

III. Accelerated Projection-Based Consensus

A. The Algorithm

Each machine i can certainly find a solution (among infinitely many) to its own highly under-determined system of equations $A_i x = b_i$, with simply $O(p^3)$ computations. We denote this initial solution by $x_i(0)$. Clearly adding any vector in the right nullspace of A_i to $x_i(0)$ will yield another viable solution. The challenge is to find vectors in the nullspaces of each of the A_i's in such a way that all the solutions for different machines coincide.

At each iteration t, the master provides the machines with an estimate of the solution, denoted by $\bar{x}(t)$. Each machine then updates its value $x_i(t)$ by projecting its difference from the estimate onto the nullspace, and taking a weighted step in that direction (which behaves as a "momentum"). Mathematically

$$x_i(t+1) = x_i(t) + \gamma P_i(\bar{x}(t) - x_i(t)),$$

where

$$P_i = I - A_i^T(A_i A_i^T)^{-1} A_i \tag{2}$$

is the projection matrix onto the nullspace of A_i (It is easy to check that $A_i P_i = 0$ and $P_i^2 = P_i$).

Although this might bear some resemblance to the block Cimmino method because of the projection matrices, APC has a much faster convergence rate than the block Cimmino method (i.e. convergence time smaller by a square root), as will be shown in Section IV. Moreover, it turns out that the block Cimmino method is in fact a special case of APC for $\gamma = 1$ (Section IV-E).

The update rule of $x_i(t+1)$ described above can be also thought of as the solution to an optimization problem with two terms: the distance from the global estimate $\bar{x}(t)$, and the distance from the previous solution $x_i(t)$. In other words, one can

Algorithm 1: APC: Accelerated Projection-Based Consensus (For Solving $Ax = b$ Distributedly).

Input: data $[A_i, b_i]$ on each machine $i = 1, \dots m$, parameters η, γ
Initialization: on each machine i, find a solution $x_i(0)$ (among infinitely many) to $A_i x = b_i$.
at the master, compute $\bar{x}(0) \leftarrow \frac{1}{m}\sum_{i=1}^m x_i(0)$
for $t = 1$ **to** T **do**
  for each machine i **parallel do**
    $x_i(t) \leftarrow x_i(t-1) + \gamma P_i(\bar{x}(t-1) - x_i(t-1))$
  end for
  at the master: $\bar{x}(t) \leftarrow \frac{\eta}{m}\sum_{i=1}^m x_i(t) + (1-\eta)\bar{x}(t-1)$
end for

show that

$$x_i(t+1) = \arg\min_{x_i} \quad \|x_i - \bar{x}(t)\|^2 + \frac{1-\gamma}{\gamma}\|x_i - x_i(t)\|^2$$
$$\text{s.t.} \quad A_i x_i = b_i$$

The second term in the objective is what distinguishes this method from the block Cimmino method. If one sets γ equal to 1 (which is the reduction to the block Cimmino method), the second term disappears altogether, and the update no longer depends on $x_i(t)$. As we will show, this can have a dramatic impact on the convergence rate.

After each iteration, the master collects the updated values $x_i(t+1)$ to form a new estimate $\bar{x}(t+1)$. A plausible choice for this is to simply take the average of the values as the new estimate, i.e., $\bar{x}(t+1) = \frac{1}{m}\sum_{i=1}^m x_i(t+1)$. This update works, and is what appears both in ADMM and in the consensus method of [19], [20]. But it turns out that it is extremely slow. Instead, we take an affine combination of the average and the previous estimate as

$$\bar{x}(t+1) = \frac{\eta}{m}\sum_{i=1}^m x_i(t+1) + (1-\eta)\bar{x}(t),$$

which introduces a one-step memory, and again behaves as a momentum.

The resulting update rule is therefore

$$x_i(t+1) = x_i(t) + \gamma P_i(\bar{x}(t) - x_i(t)), i \in [m], \tag{3a}$$

$$\bar{x}(t+1) = \frac{\eta}{m}\sum_{i=1}^m x_i(t+1) + (1-\eta)\bar{x}(t), \tag{3b}$$

which leads to Algorithm 1.

B. Convergence Analysis

We analyze the convergence of the proposed algorithm and prove that it has linear convergence (i.e. the error decays exponentially), with no additional assumption imposed. We also derive the rate of convergence explicitly.

Let us define the matrix $X \in \mathbb{R}^{n\times n}$ as

$$X \triangleq \frac{1}{m}\sum_{i=1}^m A_i^T (A_i A_i^T)^{-1} A_i. \tag{4}$$

As it will become clear soon, the condition number of this matrix predicts the behavior of the algorithm. Note that since the eigenvalues of the projection matrix P_i are all 0 and 1, for every i, the eigenvalues of X are all between 0 and 1. Denoting the eigenvalues of X by μ_i, $0 \le \mu_{\min} \triangleq \mu_n \le \cdots \le \mu_1 \triangleq \mu_{\max} \le 1$. Let us define complex quadratic polynomials $p_i(\lambda)$ characterized by γ and η as

$$p_i(\lambda;\gamma,\eta) \triangleq \lambda^2 + (-\eta\gamma(1-\mu_i) + \gamma - 1 + \eta - 1)\lambda + (\gamma-1)(\eta-1) \tag{5}$$

for $i = 1, \dots, n$. Further, define set S as the collection of pairs $\gamma \in [0, 2]$ and $\eta \in \mathbb{R}$ for which the largest magnitude solution of $p_i(\lambda) = 0$ among every i is less than 1, i.e.

$$S = \{(\gamma,\eta) \in [0,2]\times\mathbb{R} \mid \text{roots of } p_i \text{ have magnitude less than 1 for all } i\}. \tag{6}$$

The following result summarizes the convergence behavior of the proposed algorithm.

Theorem 1: Algorithm 1 converges to the true solution as fast as ρ^t converges to 0, as $t \to \infty$, for some $\rho \in (0,1)$, if and only if $(\gamma,\eta) \in S$. Furthermore, the optimal rate of convergence is

$$\rho = \frac{\sqrt{\kappa(X)}-1}{\sqrt{\kappa(X)}+1} \approx 1 - \frac{2}{\sqrt{\kappa(X)}}, \tag{7}$$

where $\kappa(X) = \frac{\mu_{\max}}{\mu_{\min}}$ is the condition number of X, and the optimal parameters (γ^*, η^*) are the solution to the following equations

$$\begin{cases} \mu_{\max}\eta\gamma = (1 + \sqrt{(\gamma-1)(\eta-1)})^2, \\ \mu_{\min}\eta\gamma = (1 - \sqrt{(\gamma-1)(\eta-1)})^2. \end{cases}$$

Proof: Let x^* be the solution of $Ax = b$. To make the analysis easier, we define error vectors with respect to x^* as $e_i(t) = x_i(t) - x^*$ for all $i = 1 \dots m$, and $\bar{e}(t) = \bar{x}(t) - x^*$, and work with these vectors. Using this notation, Eq. (3a) can be rewritten as

$$e_i(t+1) = e_i(t) + \gamma P_i(\bar{e}(t) - e_i(t)), i = 1, \dots, m.$$

Note that both x^* and $x_i(t)$ are solutions to $A_i x = b_i$. Therefore, their difference, which is $e_i(t)$, is in the nullspace of A_i, and it remains unchanged under projection onto the nullspace. As a result, $P_i e_i(t) = e_i(t)$, and we have

$$e_i(t+1) = (1-\gamma)e_i(t) + \gamma P_i \bar{e}(t), i = 1, \dots, m. \tag{8}$$

Similarly, the recursion (3b) can be expressed as

$$\bar{e}(t+1) = \frac{\eta}{m}\sum_{i=1}^m e_i(t+1) + (1-\eta)\bar{e}(t),$$

which using (8) becomes

$$\begin{aligned}\bar{e}(t+1) &= \frac{\eta}{m}\sum_{i=1}^{m}\left((1-\gamma)e_i(t)+\gamma P_i\bar{e}(t)\right)+(1-\eta)\bar{e}(t)\\ &= \frac{\eta(1-\gamma)}{m}\sum_{i=1}^{m}e_i(t)+\left(\frac{\eta\gamma}{m}\sum_{i=1}^{m}P_i+(1-\eta)I_n\right)\bar{e}(t).\end{aligned} \tag{9}$$

It is relatively easy to check that in the steady state, the recursions (8), (9) become

$$\begin{cases}P_i\bar{e}(\infty)=e_i(\infty), i=1,\dots,m\\ \bar{e}(\infty)=\frac{1}{m}\sum_{i=1}^{m}P_i\bar{e}(\infty)\end{cases}$$

which because of $\frac{1}{m}\sum_{i=1}^{m}P_i = I-\frac{1}{m}\sum_{i=1}^{m}A_i^T(A_iA_i^T)^{-1}A_i = I-X$, implies $\bar{e}(\infty)=e_1(\infty)=\cdots=e_m(\infty)=0$, if $\mu_{\min}\neq 0$.

Now let us stack up all the m vectors e_i along with the average $\bar{e}$ together, as a vector $e(t)^T=[e_1(t)^T, e_2(t)^T,\dots,e_m(t)^T,\bar{e}(t)^T]\in\mathbb{R}^{(m+1)n}$. The update rule can be expressed as:

$$\begin{bmatrix}e_1(t+1)\\ \vdots\\ e_m(t+1)\\ \bar{e}(t+1)\end{bmatrix}=\begin{bmatrix}(1-\gamma)I_{mn} & \gamma\begin{bmatrix}P_1\\ \vdots\\ P_m\end{bmatrix}\\ \frac{\eta(1-\gamma)}{m}\begin{bmatrix}I_n\dots I_n\end{bmatrix} & M\end{bmatrix}\begin{bmatrix}e_1(t)\\ \vdots\\ e_m(t)\\ \bar{e}(t)\end{bmatrix}, \tag{10}$$

where $M=\frac{\eta\gamma}{m}\sum_{i=1}^{m}P_i+(1-\eta)I_n$.

The convergence rate of the algorithm is determined by the spectral radius (largest magnitude eigenvalue) of the $(m+1)n\times(m+1)n$ block matrix in (10). The eigenvalues λ_i of this matrix are indeed the solutions to the following characteristic equation.

$$\det\begin{bmatrix}(1-\gamma-\lambda)I_{mn} & \gamma\begin{bmatrix}P_1\\ \vdots\\ P_m\end{bmatrix}\\ \frac{\eta(1-\gamma)}{m}\begin{bmatrix}I_n\dots I_n\end{bmatrix} & \frac{\eta\gamma}{m}\sum_{i=1}^{m}P_i+(1-\eta-\lambda)I_n\end{bmatrix}=0.$$

Using the Schur complement and properties of determinant, the characteristic equation can be simplified as follows.

$$\begin{aligned}0&=(1-\gamma-\lambda)^{mn}\\ &\times\det\left(\frac{\eta\gamma}{m}\sum_{i=1}^{m}P_i+(1-\eta-\lambda)I_n-\frac{\eta(1-\gamma)\gamma}{(1-\gamma-\lambda)m}\sum_{i=1}^{m}P_i\right)\\ &=(1-\gamma-\lambda)^{mn}\\ &\quad\times\det\left(\frac{\eta\gamma}{m}\left(1-\frac{1-\gamma}{1-\gamma-\lambda}\right)\sum_{i=1}^{m}P_i+(1-\eta-\lambda)I_n\right)\\ &=(1-\gamma-\lambda)^{mn}\\ &\quad\times\det\left(\frac{-\eta\gamma\lambda}{(1-\gamma-\lambda)m}\sum_{i=1}^{m}P_i+(1-\eta-\lambda)I_n\right)\\ &=(1-\gamma-\lambda)^{(m-1)n}\\ &\quad\times\det\left(-\eta\gamma\lambda\frac{\sum_{i=1}^{m}P_i}{m}+(1-\gamma-\lambda)(1-\eta-\lambda)I_n\right).\end{aligned}$$

Therefore, there are $(m-1)n$ eigenvalues equal to $1-\gamma$, and the remaining $2n$ eigenvalues are the solutions to

$$\begin{aligned}0&=\det\left(-\eta\gamma\lambda(I-X)+(1-\gamma-\lambda)(1-\eta-\lambda)I\right)\\ &=\det\left(\eta\gamma\lambda X+((1-\gamma-\lambda)(1-\eta-\lambda)-\eta\gamma\lambda)\,I\right).\end{aligned}$$

Whenever we have dropped the subscript of the identity matrix, it is of size n.

Recall that the eigenvalues of X are denoted by $\mu_i,\ \ i=1,\dots,n$. Therefore, the eigenvalues of $\eta\gamma\lambda X+((1-\gamma-\lambda)(1-\eta-\lambda)-\eta\gamma\lambda)\,I$ are $\eta\gamma\lambda\mu_i+(1-\gamma-\lambda)(1-\eta-\lambda)-\eta\gamma\lambda,\ \ i=1,\dots,n$. The above determinant can then be written as the product of the eigenvalues of the matrix inside it, as

$$0=\prod_{i=1}^{n}\eta\gamma\lambda\mu_i+(1-\gamma-\lambda)(1-\eta-\lambda)-\eta\gamma\lambda.$$

Therefore, there are two eigenvalues $\lambda_{i,1},\lambda_{i,2}$ as the solution to the quadratic equation

$$\lambda^2+(-\eta\gamma(1-\mu_i)+\gamma-1+\eta-1)\lambda+(\gamma-1)(\eta-1)=0$$

for every $i=1,\dots,n$, which will constitute the $2n$ eigenvalues. When all these eigenvalues, along with $1-\gamma$, are less than 1, the error converges to zero as ρ^t, with ρ being the largest magnitude eigenvalue (spectral radius). Therefore, Algorithm 1 converges to the true solution x^* as fast as ρ^t converges to 0, as $t\to\infty$, if and only if $(\gamma,\eta)\in S$.

The optimal rate of convergence is achieved when the spectral radius is minimum. For that to happen, all the above eigenvalues should be complex and have magnitude $|\lambda_{i,1}|=|\lambda_{i,2}|=\sqrt{(\gamma-1)(\eta-1)}=\rho$. It implies that we should have

$$(\gamma+\eta-\eta\gamma(1-\mu_i)-2)^2\leq 4(\gamma-1)(\eta-1),\forall i,$$

or equivalently

$$\begin{aligned}-2\sqrt{(\gamma-1)(\eta-1)}&\leq\gamma+\eta-\eta\gamma(1-\mu_i)\\ &\leq 2\sqrt{(\gamma-1)(\eta-1)}\end{aligned}$$

for all i. The expression in the middle is an increasing function of μ_i, and therefore for the above bounds to hold, it is enough for the lower bound to hold for the $\mu_{\min}$ and the upper bound to hold for $\mu_{\max}$, i.e.

$$\begin{cases}\gamma+\eta-\eta\gamma(1-\mu_{\max})-2=2\sqrt{(\gamma-1)(\eta-1)}\\ 2+\eta\gamma(1-\mu_{\min})-\gamma-\eta=2\sqrt{(\gamma-1)(\eta-1)}\end{cases},$$

which can be massaged and expressed as

$$\begin{cases}\mu_{\max}\eta\gamma=(1+\sqrt{(\gamma-1)(\eta-1)})^2=(1+\rho)^2\\ \mu_{\min}\eta\gamma=(1-\sqrt{(\gamma-1)(\eta-1)})^2=(1-\rho)^2\end{cases}.$$

Dividing the above two equations implies $\kappa(X) = \frac{(1+\rho)^2}{(1-\rho)^2}$, which results in the optimal rate of convergence being

$$\rho = \frac{\sqrt{\kappa(X)} - 1}{\sqrt{\kappa(X)} + 1},$$

and that concludes the proof. ■

We should remark that while in theory the optimal values of γ and η depend on the values of the smallest and largest eigenvalues of X, in practice, one will almost never compute these eigenvalues. Rather, one will use surrogate heuristics (such as using the eigenvalues of an appropriate-size random matrix) to choose the step size. (In fact, the other methods, such as distributed gradient descent and its variants, have the same issue as well.)

C. Computation and Communication Complexity

In addition to the convergence rate, or equivalently the number of iterations until convergence, one needs to consider the computational complexity per iteration. At each iteration, since $P_i = I_n - A_i^T(A_iA_i^T)^{-1}A_i$, and A_i is $p \times n$, each machine has to do the following two matrix-vector multiplications: (1) $A_i(x_i(t) - \bar{x}(t))$, which takes pn scalar multiplications, and (2) $\left(A_i^T(A_iA_i^T)^{-1}\right)$ times the vector from the previous step, which takes another np operations (the pseudoinverse $A_i^T(A_iA_i^T)^{-1}$ is computed only once). Thus the overall computational complexity of each iteration is $2pn$.

We should remark that the computation done at each machine during each iteration is essentially a projection, which has condition number one and is as numerically stable as a matrix vector multiplication can be.

Finally, the communication cost of the algorithm, per iteration, is as follows. After computing the update, each of the m machines sends an n-dimensional vector to the master, and receives back another n-dimensional vector, which is the new average.

As we will see, the per-iteration computation and communication complexity of the other algorithms are similar to APC; however, APC requires fewer iterations, because of its faster rate of convergence.

IV. Comparison With Related Methods

A. Distributed Gradient Descent (DGD)

As mentioned earlier, (1) can also be viewed as an optimization problem of the form

$$\underset{x}{\text{minimize}} \ \|Ax - b\|^2,$$

and since the objective is separable in the data, i.e. $\|Ax - b\|^2 = \sum_{i=1}^m \|A_ix - b_i\|^2$, generic distributed optimization methods such as distributed gradient descent apply well to the problem.

The regular or full gradient descent has the update rule $x(t+1) = x(t) - \alpha A^T(Ax(t) - b)$, where $\alpha > 0$ is the step size or learning rate. The distributed version of gradient descent is one in which each machine i has only a subset of the equations $[A_i, b_i]$, and computes its own part of the gradient, which is $A_i^T(A_ix(t) - b_i)$. The updates are then collectively done as:

$$x(t+1) = x(t) - \alpha \sum_{i=1}^m A_i^T(A_ix(t) - b_i). \tag{11}$$

One can show that this also has linear convergence, and the rate of convergence is

$$\rho_{\text{GD}} = \frac{\kappa(A^TA) - 1}{\kappa(A^TA) + 1} \approx 1 - \frac{2}{\kappa(A^TA)}. \tag{12}$$

We should mention that since each machine needs to compute $A_i^T(A_ix(t) - b_i)$ at each iteration t, the computational complexity per iteration is $2pn$, which is identical to that of APC.

B. Distributed Nesterov's Accelerated Gradient Descent (D-NAG)

A popular variant of gradient descent is Nesterov's accelerated gradient descent [5], which has a memory term, and works as follows:

$$y(t+1) = x(t) - \alpha \sum_{i=1}^m A_i^T(A_ix(t) - b_i), \tag{13a}$$

$$x(t+1) = (1+\beta)y(t+1) - \beta y(t). \tag{13b}$$

One can show [21] that the optimal convergence rate of this method is

$$\rho_{\text{NAG}} = 1 - \frac{2}{\sqrt{3\kappa(A^TA) + 1}}, \tag{14}$$

which is improved over the regular distributed gradient descent (one can check that $\frac{\kappa(A^TA)-1}{\kappa(A^TA)+1} \geq 1 - \frac{2}{\sqrt{3\kappa(A^TA)+1}}$).

C. Distributed Heavy-Ball Method (D-HBM)

The heavy-ball method [6], otherwise known as the gradient descent with momentum, is another accelerated variant of gradient descent as follows:

$$z(t+1) = \beta z(t) + \sum_{i=1}^m A_i^T(A_ix(t) - b_i), \tag{15a}$$

$$x(t+1) = x(t) - \alpha z(t+1). \tag{15b}$$

It can be shown [21] that the optimal rate of convergence of this method is

$$\rho_{\text{HBM}} = \frac{\sqrt{\kappa(A^TA)} - 1}{\sqrt{\kappa(A^TA)} + 1} \approx 1 - \frac{2}{\sqrt{\kappa(A^TA)}}, \tag{16}$$

which is further improved over DGD and D-NAG ($\frac{\kappa(A^TA)-1}{\kappa(A^TA)+1} \geq 1 - \frac{2}{\sqrt{3\kappa(A^TA)+1}} \geq \frac{\sqrt{\kappa(A^TA)}-1}{\sqrt{\kappa(A^TA)}+1}$). This is similar to, but not the same as, the rate of convergence of APC. The difference is that the condition number of $A^TA = \sum_{i=1}^m A_i^TA_i$ is replaced with the condition number of $X = \sum_{i=1}^m A_i^T\left(A_iA_i^T\right)^{-1}A_i$ in APC. Given its structure as the sum of projection matrices, one may speculate that X has a much better condition number than A^TA.

TABLE I
A SUMMARY OF THE CONVERGENCE RATES OF DIFFERENT METHODS. DGD: DISTRIBUTED GRADIENT DESCENT, D-NAG: DISTRIBUTED NESTEROV'S ACCELERATED GRADIENT DESCENT, D-HBM: DISTRIBUTED HEAVY-BALL METHOD, MOU ET AL: CONSENSUS ALGORITHM OF [20], B-CIMMINO: BLOCK CIMMINO METHOD, APC: ACCELERATED PROJECTION-BASED CONSENSUS. THE SMALLER THE CONVERGENCE RATE IS, THE FASTER IS THE METHOD. NOTE THAT $\rho_{\rm GD} \geq \rho_{\rm NAG} \geq \rho_{\rm HBM}$ AND $\rho_{\rm Mou} \geq \rho_{\rm Cim} \geq \rho_{\rm APC}$

DGD	D-NAG	D-HBM	Mou et al.	B-Cimmino	APC (proposed)
$\frac{\kappa(A^TA)-1}{\kappa(A^TA)+1}$ $\approx 1-\frac{2}{\kappa(A^TA)}$	$1-\frac{2}{\sqrt{3\kappa(A^TA)+1}}$	$\frac{\sqrt{\kappa(A^TA)}-1}{\sqrt{\kappa(A^TA)}+1}$ $\approx 1-\frac{2}{\sqrt{\kappa(A^TA)}}$	$1-\mu_{\min}(X)$	$\frac{\kappa(X)-1}{\kappa(X)+1}$ $\approx 1-\frac{2}{\kappa(X)}$	$\frac{\sqrt{\kappa(X)}-1}{\sqrt{\kappa(X)}+1}$ $\approx 1-\frac{2}{\sqrt{\kappa(X)}}$

TABLE II
A COMPARISON BETWEEN THE CONDITION NUMBERS OF A^TA AND X FOR SOME EXAMPLES. m IS THE NUMBER OF MACHINES/PARTITIONS. THE CONDITION NUMBER OF X IS TYPICALLY MUCH SMALLER (BETTER). REMARKABLY, THE DIFFERENCE IS EVEN MORE PRONOUNCED WHEN A HAS NON-ZERO MEAN

	$\kappa(A^TA)$	$\kappa(X)$
100×100 $\mathcal{N}(0,1)$ (GAUSSIAN)	7×10^4	2×10^4 $(m=2)$ 4×10^4 $(m=5)$ 5×10^4 $(m=10)$ 6×10^4 $(m=20)$
100×100 $\mathcal{N}(10,1)$ (NON-ZERO MEAN)	2×10^8	4×10^6 $(m=2)$ 1×10^7 $(m=5)$ 2×10^7 $(m=10)$ 4×10^7 $(m=20)$
100×100 $\mathcal{N}(0,10^2)$ (HIGH VARIANCE)	8×10^5	2×10^5 $(m=2)$ 5×10^5 $(m=5)$ 6×10^5 $(m=10)$ 7×10^5 $(m=20)$
200×100 $\mathcal{N}(0,1)$ (TALL)	3×10^1	1×10^0 $(m=2)$ 1×10^1 $(m=5)$ 2×10^1 $(m=10)$ 2×10^1 $(m=20)$
100×100 $\exp(10)$ (EXPONENTIAL)	9×10^5	1×10^4 $(m=2)$ 4×10^4 $(m=5)$ 1×10^5 $(m=10)$ 2×10^5 $(m=20)$
100×100 $\text{stable}(0.5,0.5,1,0)$ (HEAVY TAIL)	3×10^{15}	1×10^7 $(m=2)$ 3×10^7 $(m=5)$ 3×10^7 $(m=10)$ 3×10^7 $(m=20)$
REAL EXAMPLE: QC324 (324×324)	2×10^7	1×10^5 $(m=2)$ 3×10^5 $(m=4)$ 6×10^5 $(m=9)$ 6×10^5 $(m=18)$ 8×10^5 $(m=81)$
REAL EXAMPLE: ORSIRR 1 (1030×1030)	6×10^9	4×10^7 $(m=2)$ 5×10^7 $(m=5)$ 5×10^7 $(m=10)$ 6×10^7 $(m=103)$ 6×10^7 $(m=206)$
REAL EXAMPLE: ASH608 (608×188)	11×10^0	8×10^0 $(m=8)$ 10×10^0 $(m=16)$ 10×10^0 $(m=32)$ 11×10^0 $(m=76)$

Indeed, our experiments with random, as well as real, data sets suggest that this is the case and that the condition number of X is often significantly better (see Table II).

D. Alternating Direction Method of Multipliers (ADMM)

Alternating Direction Method of Multipliers (more specifically, consensus ADMM [7], [12]), is another generic method for solving optimization problems with separable cost function $f(x)=\sum_{i=1}^m f_i(x)$ distributedly, by defining additional local variables. Each machine i holds local variables $x_i(t)\in\mathbb{R}^n$ and $y_i(t)\in\mathbb{R}^n$, and the master's value is $\bar{x}(t)\in\mathbb{R}^n$, for any time t. For $f_i(x)=\frac{1}{2}\|A_ix-b_i\|^2$, the update rule of ADMM simplifies to

$$x_i(t+1)=(A_i^TA_i+\xi I_n)^{-1}(A_i^Tb_i-y_i(t)+\xi\bar{x}(t)),\quad i\in[m] \tag{17a}$$

$$\bar{x}(t+1)=\frac{1}{m}\sum_{i=1}^m x_i(t+1) \tag{17b}$$

$$y_i(t+1)=y_i(t)+\xi(x_i(t+1)-\bar{x}(t+1)),i\in[m] \tag{17c}$$

It turns out that this method is very slow (and often unstable) in its native form for the application in hand. One can check that when system (1) has a solution, all the y_i variables converge to zero in steady state. Therefore, setting y_i's to zero can speed up the convergence significantly. We use this modified version in Section VI, to compare with.

We should also note that the computational complexity of ADMM is $O(pn)$ per iteration (the inverse is computed using matrix inversion lemma), which is again the same as that of gradient-type methods and APC.

E. Block Cimmino Method

The Block Cimmino method [14]–[16], which is a parallel method specifically for solving linear systems of equations, is perhaps the closest algorithm in spirit to APC. It is, in a way, a distributed implementation of the so-called Kaczmarz method [18]. The convergence of the Cimmino method is slower by an order in comparison with APC (its convergence time is the square of that of APC), and it turns out that APC includes this method as a special case when $\gamma=1$.

The block Cimmino method is the following:

$$r_i(t)=A_i^+(b_i-A_i\bar{x}(t)),i\in[m] \tag{18a}$$

$$\bar{x}(t+1)=\bar{x}(t)+\nu\sum_{i=1}^m r_i(t), \tag{18b}$$

where $A_i^+=A_i^T(A_iA_i^T)^{-1}$ is the pseudoinverse of A_i.

Proposition 2: The APC method (Algorithm 1) includes the block Cimmino method as a special case for $\gamma = 1$.

Proof: When $\gamma = 1$, Eq. (3a) becomes

$$\begin{aligned} x_i(t+1) &= x_i(t) - P_i\left(x_i(t) - \bar{x}(t)\right) \\ &= x_i(t) - \left(I - A_i^T(A_iA_i^T)^{-1}A_i\right)\left(x_i(t) - \bar{x}(t)\right) \\ &= \bar{x}(t) + A_i^T(A_iA_i^T)^{-1}A_i(x_i - \bar{x}(t)) \\ &= \bar{x}(t) + A_i^T(A_iA_i^T)^{-1}(b_i - A_i\bar{x}(t)) \end{aligned}$$

In the last equation, we used the fact that x_i is always a solution to $A_ix = b_i$. Notice that the above equation is no longer an "update" in the usual sense, i.e., $x_i(t+1)$ does not depend on $x_i(t)$ directly. This can be further simplified using the pseudoinverse of A_i, $A_i^+ = A_i^T(A_iA_i^T)^{-1}$ as

$$x_i(t+1) = \bar{x}(t) + A_i^+(b_i - A_i\bar{x}(t)).$$

It is then easy to see from the Cimmino's equation (18a) that

$$r_i(t) = x_i(t+1) - \bar{x}(t).$$

Therefore, the update (18b) can be expressed as

$$\begin{aligned} \bar{x}(t+1) &= \bar{x}(t) + \nu\sum_{i=1}^m r_i(t) \\ &= \bar{x}(t) + \nu\sum_{i=1}^m \left(x_i(t+1) - \bar{x}(t)\right) \\ &= (1 - m\nu)\bar{x}(t) + \nu\sum_{i=1}^m x_i(t+1), \end{aligned}$$

which is nothing but the same update rule as in (3b) with $\eta = m\nu$. ■

It is not hard to show that optimal rate of convergence of the Cimmino method is

$$\rho_{\text{Cim}} = \frac{\kappa(X) - 1}{\kappa(X) + 1} \approx 1 - \frac{2}{\kappa(X)}, \tag{19}$$

which is slower (by an order) than that of APC ($\frac{\sqrt{\kappa(X)}-1}{\sqrt{\kappa(X)}+1} \approx 1 - \frac{2}{\sqrt{\kappa(X)}}$).

F. Consensus Algorithm of Mou et el.

As mentioned earlier, a projection-based consensus algorithm for solving linear systems over a network was recently proposed by Mou *et al.* [19], [20]. For the master-worker setting studied in this paper, the corresponding network would be a clique, and the algorithm reduces to

$$x_i(t+1) = x_i(t) + P_i\left(\frac{1}{m}\left(\sum_{j=1}^m x_j(t)\right) - x_i(t)\right), i \in [m], \tag{20}$$

which is transparently equivalent to APC with $\gamma = \eta = 1$:

$$\begin{aligned} x_i(t+1) &= x_i(t) + P_i(\bar{x}(t) - x_i(t)), i \in [m], \\ \bar{x}(t+1) &= \frac{1}{m}\sum_{i=1}^m x_i(t+1), \end{aligned}$$

It is straightforward to show that the rate of convergence in this case is

$$\rho_{\text{Mou}} = 1 - \mu_{\min}(X) \tag{21}$$

which is much slower than the block Cimmino method and APC. One can easily check that

$$1 - \mu_{\min}(X) \geq \frac{\kappa(X) - 1}{\kappa(X) + 1} \geq \frac{\sqrt{\kappa(X)} - 1}{\sqrt{\kappa(X)} + 1}.$$

Even though this algorithm is slow, it is useful for applications where a fully-distributed (networked) solution is desired. Another networked algorithm for solving a least-squares problem has been recently proposed in [22].

A summary of the convergence rates of all the related methods discussed in this section is provided in Table I.

V. Underdetermined System

In this section, we consider the case when $N < n$ and $\text{rank}(A) = N$, i.e., the system is underdetermined and there are infinitely many solutions. We prove that in this case, each machine still converges to "a" (global) solution, and further, all the machines converge to the same solution. The convergence is again linear (i.e. the error decays exponentially fast), and the rate of convergence is similar to the previous case.

Recall that the matrix $X \in \mathbb{R}^{n\times n}$ defined earlier can be written as

$$\begin{aligned} X &= \frac{1}{m}\sum_{i=1}^m A_i^T(A_iA_i^T)^{-1}A_i \\ &= \frac{1}{m}A^T\begin{bmatrix}(A_1A_1^T)^{-1} & & \\ & \ddots & \\ & & (A_mA_m^T)^{-1}\end{bmatrix}A \end{aligned}$$

which is singular in this case.

We define a new matrix $Y \in \mathbb{R}^{N\times N}$

$$Y \triangleq \frac{1}{m}AA^T\begin{bmatrix}(A_1A_1^T)^{-1} & & \\ & \ddots & \\ & & (A_mA_m^T)^{-1}\end{bmatrix} \tag{22}$$

which has the same nonzero eigenvalues as X.

Theorem 3: Suppose $N < n$ and $\text{rank}(A) = N$. Each one of $x_1(t), \ldots, x_m(t), \bar{x}(t)$ in Algorithm 1 converges to a solution as fast as ρ^t converges to 0, as $t \to \infty$, for some $\rho \in (0, 1)$, if and only if $(\gamma, \eta) \in S$. Furthermore, the solutions converged to are the same. The optimal rate of convergence is

$$\rho = \frac{\sqrt{\kappa(Y)} - 1}{\sqrt{\kappa(Y)} + 1} \approx 1 - \frac{2}{\sqrt{\kappa(Y)}}, \tag{23}$$

where $\kappa(Y) = \frac{\mu_{\max}}{\mu_{\min}}$ is the condition number of Y, and the optimal parameters (γ^*, η^*) are the solution to the following equations

$$\begin{cases}\mu_{\max}\eta\gamma = (1 + \sqrt{(\gamma-1)(\eta-1)})^2, \\ \mu_{\min}\eta\gamma = (1 - \sqrt{(\gamma-1)(\eta-1)})^2.\end{cases}$$

Proof: Let x^* be a solution to $Ax = b$. We define error vectors $e_i(t) = x_i(t) - x^*$ for all $i = 1 \dots m$, and $\bar{e}(t) = \bar{x}(t) - x^*$, as before, but this time show that $Ae_i(t) \to 0$ and $A\bar{e}(t) \to 0$. Recursion (3a) can be rewritten as

$$e_i(t+1) = e_i(t) + \gamma P_i(\bar{e}(t) - e_i(t)), i = 1, \dots, m,$$

as before. Since both x^* and $x_i(t)$ are solutions to $A_i x = b_i$, their difference $e_i(t)$ is in the nullspace of A_i, and it remains unchanged under projection onto the nullspace. As a result, $P_i e_i(t) = e_i(t)$, and we have

$$e_i(t+1) = (1-\gamma)e_i(t) + \gamma P_i \bar{e}(t), i = 1, \dots, m. \quad (24)$$

Similarly, the recursion (3b) can be expressed as

$$\begin{aligned}\bar{e}(t+1) &= \frac{\eta}{m}\sum_{i=1}^{m} e_i(t+1) + (1-\eta)\bar{e}(t) \\ &= \frac{\eta}{m}\sum_{i=1}^{m}((1-\gamma)e_i(t) + \gamma P_i\bar{e}(t)) + (1-\eta)\bar{e}(t) \\ &= \frac{\eta(1-\gamma)}{m}\sum_{i=1}^{m} e_i(t) + \left(\frac{\eta\gamma}{m}\sum_{i=1}^{m} P_i + (1-\eta)I_n\right)\bar{e}(t),\end{aligned}$$

as before.

Multiplying the recursions by A, we have

$$Ae_i(t+1) = (1-\gamma)Ae_i(t) + \gamma AP_i\bar{e}(t), i = 1, \dots, m,$$

and

$$\begin{aligned}A\bar{e}(t+1) = \frac{\eta(1-\gamma)}{m}\sum_{i=1}^{m} Ae_i(t) \\ + \left(\frac{\eta\gamma}{m}\sum_{i=1}^{m} AP_i + (1-\eta)A\right)\bar{e}(t)\end{aligned}$$

Note that $P_i = I_n - A_i^T(A_iA_i^T)^{-1}A_i$, and we can express A_i as $A_i = \begin{bmatrix}0_{p\times p} \dots I_p \dots 0_{p\times p}\end{bmatrix}A = E_iA$, where E_i is a $p \times N$ matrix with an identity at its i-th block and zero everywhere else. Therefore, we have $AP_i = A - AA_i^T(A_iA_i^T)^{-1}E_iA = (I_N - AA_i^T(A_iA_i^T)^{-1}E_i)A$, and the recursions become

$$\begin{aligned}Ae_i(t+1) = (1-\gamma)Ae_i(t) \\ + \gamma(I_N - AA_i^T(A_iA_i^T)^{-1}E_i)A\bar{e}(t),\end{aligned}$$

for $i = 1, \dots, m$, and

$$\begin{aligned}A\bar{e}(t+1) = \frac{\eta(1-\gamma)}{m}\sum_{i=1}^{m} Ae_i(t) \\ + \left(\frac{\eta\gamma}{m}\sum_{i=1}^{m}(I_N - AA_i^T(A_iA_i^T)^{-1}E_i) + (1-\eta)I_N\right)A\bar{e}(t)\end{aligned}$$

Stacking up all the m vectors Ae_i along with $A\bar{e}$, as an $(m+1)N$-dimensional vector, results in

$$\begin{bmatrix}Ae_1(t+1)\\ \vdots \\ Ae_m(t+1)\\ A\bar{e}(t+1)\end{bmatrix} = \begin{bmatrix}(1-\gamma)I_{mN} & \gamma\begin{bmatrix}P_1'\\ \vdots \\ P_m'\end{bmatrix}\\ \frac{\eta(1-\gamma)}{m}\begin{bmatrix}I_N \dots I_N\end{bmatrix} & M'\end{bmatrix}\begin{bmatrix}Ae_1(t)\\ \vdots \\ Ae_m(t)\\ A\bar{e}(t)\end{bmatrix},$$

where $M' = \frac{\eta\gamma}{m}\sum_{i=1}^{m} P_i' + (1-\eta)I_N$ and $P_i' = I_N - AA_i^T(A_iA_i^T)^{-1}E_i$.

The convergence rate of the algorithm is determined by the spectral radius (largest magnitude eigenvalue) of this $(m+1)N \times (m+1)N$ matrix. The eigenvalues λ_i of this matrix are the solutions to the following characteristic equation.

$$\det\begin{bmatrix}(1-\gamma-\lambda)I_{mN} & \gamma\begin{bmatrix}P_1'\\ \vdots \\ P_m'\end{bmatrix}\\ \frac{\eta(1-\gamma)}{m}\begin{bmatrix}I_N \dots I_N\end{bmatrix} & \frac{\eta\gamma}{m}\sum_{i=1}^{m} P_i' + (1-\eta-\lambda)I_N\end{bmatrix} = 0.$$

Similar as in the proof of Theorem 1, using the Schur complement and properties of determinant, the characteristic equation can be simplified as follows.

$$\begin{aligned}0 &= (1-\gamma-\lambda)^{mN} \\ &\times \det\left(\frac{\eta\gamma}{m}\sum_{i=1}^{m} P_i' + (1-\eta-\lambda)I_N - \frac{\eta(1-\gamma)\gamma}{(1-\gamma-\lambda)m}\sum_{i=1}^{m} P_i'\right) \\ &= (1-\gamma-\lambda)^{mN} \\ &\quad \times \det\left(\frac{\eta\gamma}{m}\left(1 - \frac{1-\gamma}{1-\gamma-\lambda}\right)\sum_{i=1}^{m} P_i' + (1-\eta-\lambda)I_N\right) \\ &= (1-\gamma-\lambda)^{mN} \\ &\quad \times \det\left(\frac{-\eta\gamma\lambda}{(1-\gamma-\lambda)m}\sum_{i=1}^{m} P_i' + (1-\eta-\lambda)I_N\right) \\ &= (1-\gamma-\lambda)^{(m-1)N} \\ &\quad \times \det\left(-\eta\gamma\lambda\frac{\sum_{i=1}^{m} P_i'}{m} + (1-\gamma-\lambda)(1-\eta-\lambda)I_N\right).\end{aligned}$$

Note that $\frac{1}{m}\sum_{i=1}^{m} P_i' = I_N - \frac{1}{m}\sum_{i=1}^{m} AA_i^T(A_iA_i^T)^{-1}E_i = I_N - [AA_1^T(A_1A_1^T)^{-1}, \dots, AA_m^T(A_mA_m^T)^{-1}] = I_N - Y$. There are $(m-1)N$ eigenvalues equal to $1-\gamma$, and the remaining $2N$ eigenvalues are the solutions to

$$\begin{aligned}0 &= \det\left(-\eta\gamma\lambda(I-Y) + (1-\gamma-\lambda)(1-\eta-\lambda)I\right) \\ &= \det\left(\eta\gamma\lambda Y + ((1-\gamma-\lambda)(1-\eta-\lambda) - \eta\gamma\lambda)\, I\right).\end{aligned}$$

Notice that this is exactly the same as the one in the proof of Theorem 1, with X replaced with Y.

It follows that the $Ae_1(t), \dots, Ae_m(t), A\bar{e}(t)$ converge to zero as fast as ρ^t if and only if $(\gamma, \eta) \in S$, and the optimal rate

of convergence is

$$\rho = \frac{\sqrt{\kappa(Y)} - 1}{\sqrt{\kappa(Y)} + 1}.$$

Convergence of $Ae_1(t), \ldots, Ae_m(t), A\bar{e}(t)$ to zero means that each machine and the master converge to a solution, but the solutions reached may not be the same. What remains to show is that the only steady state is the "consensus steady state." From (3), it is easy to see that the steady state $x_1(\infty), \ldots, x_m(\infty), \bar{x}(\infty)$ satisfies the following equation.

$$\begin{cases} P_i(\bar{x}(\infty) - x_i(\infty)) = 0,\ i \in [m] \\ \bar{x}(\infty) = \frac{1}{m}\sum_{i=1}^m x_i(\infty) \end{cases} \tag{25}$$

which can be written in a matrix form as

$$\begin{bmatrix} P_1 & & & -P_1 \\ & \ddots & & \vdots \\ & & P_m & -P_m \\ -\frac{I_n}{m} & \ldots & -\frac{I_n}{m} & I_n \end{bmatrix} \begin{bmatrix} x_1(\infty) \\ \vdots \\ x_m(\infty) \\ \bar{x}(\infty) \end{bmatrix} = 0. \tag{26}$$

Notice that for any $v \in \mathbb{R}^n$, the vector $\begin{bmatrix} v^T & \ldots & v^T & v^T \end{bmatrix}^T$ is a solution to this equation, which corresponds to a consensus steady state $x_1(\infty) = \cdots = x_m(\infty) = \bar{x}(\infty) = v$. Therefore, the nullspace of the above matrix is at least n dimensional, or in other words, it has n zero eigenvalues. We will argue that this matrix has only n zero eigenvalues, and therefore any steady-state solution must be a consensus. To find the eigenvalues λ_i we have to solve the following characteristic equation.

$$\det \begin{bmatrix} P_1 - \lambda I & & & -P_1 \\ & \ddots & & \vdots \\ & & P_m - \lambda I & -P_m \\ -\frac{I}{m} & \ldots & -\frac{I}{m} & (1-\lambda)I \end{bmatrix} = 0.$$

Once again, using the Schur complement we have

$$0 = \left(\prod_{i=1}^m \det(P_i - \lambda I)\right) \times \det\left((1-\lambda)I - \frac{1}{m}\sum_{i=1}^m (P_i - \lambda I)^{-1} P_i\right)$$

Note that $(P_i - \lambda I)^{-1} P_i = \frac{1}{1-\lambda} P_i$. Therefore, we can write

$$\begin{aligned} 0 &= \left(\prod_{i=1}^m \det(P_i - \lambda I)\right) \det\left((1-\lambda)I - \frac{1}{m}\sum_{i=1}^m \frac{1}{1-\lambda} P_i\right) \\ &= \left(\prod_{i=1}^m \left((-\lambda)^p (1-\lambda)^{n-p}\right)\right) \cdot \det\left((1-\lambda)I - \frac{1}{m}\sum_{i=1}^m \frac{1}{1-\lambda} P_i\right) \\ &= (-\lambda)^N (1-\lambda)^{mn-N} \det\left((1-\lambda)I - \frac{1}{m}\sum_{i=1}^m \frac{1}{1-\lambda} P_i\right) \end{aligned}$$

because P_i has p zero eigenvalues and $n-p$ one eigenvalues. Using the properties of determinant, we further have

$$\begin{aligned} 0 &= (-\lambda)^N (1-\lambda)^{(m-1)n-N} \det\left((1-\lambda)^2 I - \frac{1}{m}\sum_{i=1}^m P_i\right) \\ &= (-\lambda)^N (1-\lambda)^{(m-1)n-N} \det\left((1-\lambda)^2 I - (I - X)\right) \\ &= (-\lambda)^N (1-\lambda)^{(m-1)n-N} \det\left((\lambda^2 - 2\lambda) I + X\right) \\ &= (-\lambda)^N (1-\lambda)^{(m-1)n-N} \prod_{i=1}^m \left(\lambda^2 - 2\lambda + \mu_i\right) \end{aligned}$$

Therefore, the $(m+1)n$ eigenvalues are as follows: N zero eigenvalues, $(m-1)n - N$ eigenvalues at 1, and the remaining $2n$ are $1 \pm \sqrt{1-\mu_i}$. Note that in the underdetermined case, X has $n - N$ zero eigenvalues. Therefore, $n - N$ of $1 \pm \sqrt{1-\mu_i}$ are zero. As a result, the overall number of zero eigenvalues is $N + (n - N) = n$. This implies that the nullity of the matrix in (26) is n and any steady-state solution must be a consensus solution, which completes the proof. ■

VI. Experimental Results

In this section, we evaluate the proposed method (APC) by comparing it with the other distributed methods discussed throughout the paper, namely DGD, D-NAG, D-HBM, modified ADMM, and block Cimmino methods. We use randomly-generated problems as well as real-world ones form the National Institute of Standards and Technology (NIST) repository, *Matrix Market* [23].

We first compare the rates of convergence of the algorithms ρ, which is the spectral radius of the iteration matrix. To distinguish the differences, it is easier to compare the convergence times, which is defined as $T = \frac{1}{-\log \rho}$ ($\approx \frac{1}{1-\rho}$). We tune the parameters in all of the methods to their optimal values, to make the comparison between the methods fair. Also as mentioned before, all the algorithms have the same per-iteration computation and communication complexity. Table III shows the values of the convergence times for a number of synthetic and real-world problems with different sizes. It can be seen that APC has a much faster convergence, often by orders of magnitude. As expected from the analysis, the APC's closest competitor is the distributed heavy-ball method. Notably, in randomly-generated problems, when the mean is not zero, the gap is much larger.

To further verify the performance of the proposed algorithm, we also run all the algorithms on multiple problems, and observe the actual decay of the error. Fig. 2 shows the relative error (the distance from the true solution, divided by the true solution, in ℓ_2 norm) for all the methods, on two examples from the repository. Again, to make the comparison fair, all the methods have been tuned to their optimal parameters. As one can see, APC outperforms the other methods by a wide margin, which is consistent with the order-of-magnitude differences in the convergence times of Table III. We should also remark that initialization does not seem to affect the convergence behavior of our algorithm. Lastly, we should mention that our experiments on cases where there are missing updates ("straggler" machines) indicate that

TABLE III
A COMPARISON BETWEEN THE OPTIMAL CONVERGENCE TIME T ($= \frac{1}{-\log \rho}$) OF DIFFERENT METHODS ON REAL AND SYNTHETIC EXAMPLES. BOLDFACE VALUES SHOW THE SMALLEST CONVERGENCE TIME. QC324: MODEL OF H_2^+ IN AN ELECTROMAGNETIC FIELD. ORSIRR 1: OIL RESERVOIR SIMULATION. ASH608: ORIGINAL HARWELL SPARSE MATRIX TEST COLLECTION

	DGD	D-NAG	D-HBM	M-ADMM	B-CIMMINO	APC
QC324 (324×324)	1.22×10^{7}	4.28×10^{3}	2.47×10^{3}	1.07×10^{7}	3.10×10^{5}	$\mathbf{3.93 \times 10^{2}}$
ORSIRR 1 (1030×1030)	2.98×10^{9}	6.68×10^{4}	3.86×10^{4}	2.08×10^{8}	2.69×10^{7}	$\mathbf{3.67 \times 10^{3}}$
ASH608 (608×188)	5.67×10^{0}	2.43×10^{0}	1.64×10^{0}	1.28×10^{1}	4.98×10^{0}	$\mathbf{1.53 \times 10^{0}}$
STANDARD GAUSSIAN (500×500)	1.76×10^{7}	5.14×10^{3}	2.97×10^{3}	1.20×10^{6}	1.46×10^{7}	$\mathbf{2.70 \times 10^{3}}$
NONZERO-MEAN GAUSSIAN (500×500)	2.22×10^{10}	1.82×10^{5}	1.05×10^{5}	8.62×10^{8}	9.29×10^{8}	$\mathbf{2.16 \times 10^{4}}$
STANDARD TALL GAUSSIAN (1000×500)	1.58×10^{1}	4.37×10^{0}	2.78×10^{0}	4.49×10^{1}	1.13×10^{1}	$\mathbf{2.34 \times 10^{0}}$
STANDARD FAT GAUSSIAN (400×500)	1.37×10^{2}	1.38×10^{1}	8.26×10^{0}	3.17×10^{2}	1.14×10^{2}	$\mathbf{7.54 \times 10^{0}}$

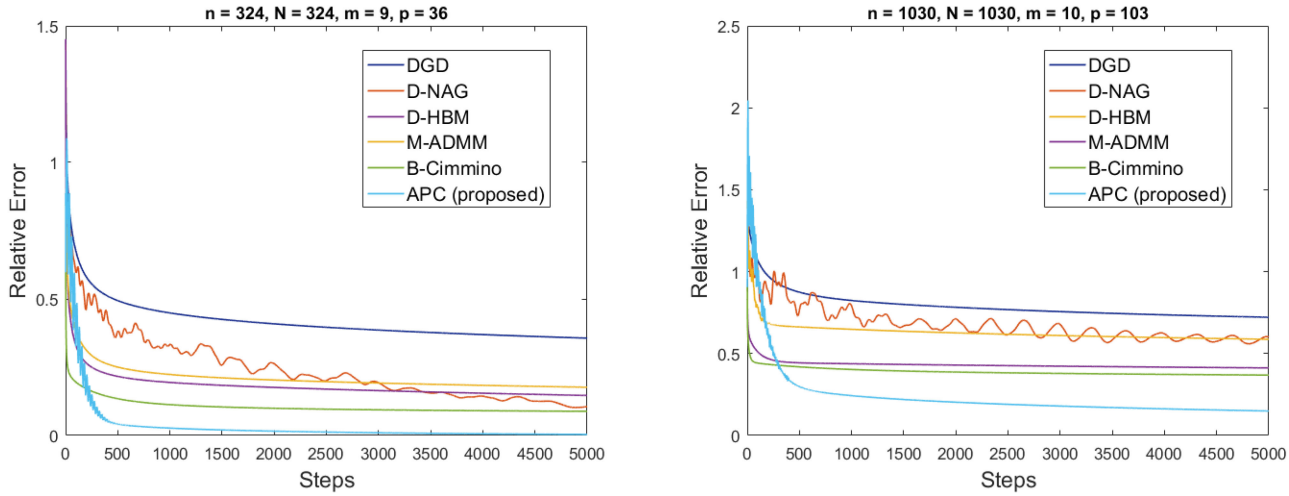

Fig. 2. The decay of the error for different distributed algorithms, on two real problems from Matrix Market [23] (QC324: Model of H_2^+ in an Electromagnetic Field, and ORSIRR 1: Oil reservoir simulation). $n = \#$ of variables, $N = \#$ of equations, $m = \#$ of workers, $p = \#$ of equations per worker.

APC is at least as robust as the other algorithms to these effects, and the convergence curves looks qualitatively the same as in Fig. 2.

VII. A DISTRIBUTED PRECONDITIONING TO IMPROVE GRADIENT-BASED METHODS

The noticeable similarity between the optimal convergence rate of APC ($\frac{\sqrt{\kappa(X)}-1}{\sqrt{\kappa(X)}+1}$) and that of D-HBM ($\frac{\sqrt{\kappa(A^TA)}-1}{\sqrt{\kappa(A^TA)}+1}$) suggests that there might be a connection between the two. It turns out that there is, and we propose a *distributed preconditioning* for D-HBM, which makes it achieve the same convergence rate as APC. The algorithm works as follows.

Prior to starting the iterative process, each machine i can premultiply its own set of equations $A_i x = b_i$ by $(A_i A_i^T)^{-1/2}$, which can be done in parallel (locally) with $O(p^2 n)$ operations. This transforms the global system of equations $Ax = b$ to a new one $Cx = d$, where

$$C = \begin{bmatrix} (A_1 A_1^T)^{-1/2} A_1 \\ \vdots \\ (A_m A_m^T)^{-1/2} A_m \end{bmatrix},$$

and

$$d = \begin{bmatrix} (A_1 A_1^T)^{-1/2} b_1 \\ \vdots \\ (A_m A_m^T)^{-1/2} b_m \end{bmatrix}.$$

The new system can then be solved using distributed heavy-ball method, which will achieve the same rate of convergence as APC, i.e. $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ where $\kappa = \kappa(C^T C) = \kappa(X)$.

VIII. Conclusion

We considered the problem of solving a large-scale system of linear equations by a taskmaster with the help of a number of computing machines/cores, in a distributed way. We proposed an accelerated projection-based consensus algorithm for this problem, and fully analyzed its convergence rate. Analytical and experimental comparisons with the other known distributed methods confirm significantly faster convergence of the proposed scheme. Finally, our analysis suggested a novel distributed preconditioning for improving the convergence of the distributed heavy-ball method to achieve the same theoretical performance as the proposed consensus-based method.

We should finally remark that while the setting studied in this paper was a master-workers one, the same algorithm can be implemented in a networked setting where there is no central collector/master, using a "distributed averaging" approach ([24], [25]).

References

[1] N. Azizan-Ruhi, F. Lahouti, S. Avestimehr, and B. Hassibi, "Distributed solution of large-scale linear systems via accelerated projection-based consensus," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Apr. 2018, pp. 6358–6362.

[2] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Proc. Advances Neural Inf. Process. Syst.*, 2010, pp. 2595–2603.

[3] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Advances Neural Inf. Process. Syst.*, 2011, pp. 693–701.

[4] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM J. Optim.*, vol. 26, no. 3, pp. 1835–1854, 2016.

[5] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," *Sov. Math. Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[6] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Comput. Math. Math. Phys.*, vol. 4, no. 5, pp. 1–17, 1964.

[7] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[8] B. He and X. Yuan, "On the $O(1/n)$ convergence rate of the Douglas-Rachford alternating direction method," *SIAM J. Numer. Anal.*, vol. 50, no. 2, pp. 700–709, 2012.

[9] W. Deng and W. Yin, "On the global and linear convergence of the generalized alternating direction method of multipliers," *J. Scientif. Comput.*, vol. 66, no. 3, pp. 889–916, 2016.

[10] R. Zhang and J. T. Kwok, "Asynchronous distributed ADMM for consensus optimization," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1701–1709.

[11] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Puschel, "D-ADMM: A communication-efficient distributed algorithm for separable optimization," *IEEE Trans. Signal Process.*, vol. 61, no. 10, pp. 2718–2723, May 2013.

[12] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in decentralized consensus optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 7, pp. 1750–1761, Apr. 2014.

[13] L. Majzoubi and F. Lahouti, "Analysis of distributed ADMM algorithm for consensus optimization in presence of error," in *Proc. IEEE Conf. Audio, Speech, Signal Process.*, Mar. 2016, pp. 4831–4835.

[14] I. S. Duff, R. Guivarch, D. Ruiz, and M. Zenadi, "The augmented block Cimmino distributed method," *SIAM J. Scientif. Comput.*, vol. 37, no. 3, pp. A1248–A1269, 2015.

[15] F. Sloboda, "A projection method of the Cimmino type for linear algebraic systems," *Parallel Comput.*, vol. 17, no. 4/5, pp. 435–442, 1991.

[16] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, "A block projection method for sparse matrices," *SIAM J. Scientif. Statist. Comput.*, vol. 13, no. 1, pp. 47–70, 1992.

[17] R. Bramley and A. Sameh, "Row projection methods for large nonsymmetric linear systems," *SIAM J. Scientif. Statist. Comput.*, vol. 13, no. 1, pp. 168–193, 1992.

[18] S. Kaczmarz, "Angenäherte auflösung von systemen linearer gleichungen," *Bulletin Int. de lAcademie Polonaise des Sci. et des Lettres*, vol. 35, pp. 355–357, 1937.

[19] J. Liu, S. Mou, and A. S. Morse, "An asynchronous distributed algorithm for solving a linear algebraic equation," in *Proc. 52nd IEEE Annu. Conf. Decis, Control*, 2013, pp. 5409–5414.

[20] S. Mou, J. Liu, and A. S. Morse, "A distributed algorithm for solving a linear algebraic equation," *IEEE Trans. Autom. Control*, vol. 60, no. 11, pp. 2863–2878, Nov. 2015.

[21] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM J. Optim.*, vol. 26, no. 1, pp. 57–95, 2016.

[22] X. Wang, J. Zhou, S. Mou, and M. J. Corless, "A distributed algorithm for least squares solutions," *IEEE Trans. Autom. Control*, 2019, doi: 10.1109/TAC.2019.2894588.

[23] "Matrix market," 2007. [Online]. Available: http://math.nist.gov/MatrixMarket/. Accessed: May 2017.

[24] J. N. Tsitsiklis, "Problems in decentralized decision making and computation," M.S. thesis, Laboratory Inf. Decis. Syst., Massachusetts Inst. Technol., Cambridge, MA, USA, 1984.

[25] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Syst. Control Lett.*, vol. 53, no. 1, pp. 65–78, 2004.

Navid Azizan Ruhi (S'15) received the B.S. degree form Sharif University of Technology, Tehran, Iran, and the M.S. degree from the University of Southern California, Los Angeles, CA, USA, in 2013 and 2015, respectively, both in electrical engineering. He is currently working toward the Ph.D. degree in computing and mathematical sciences with the California Institute of Technology, Los Angeles, CA, USA. His research interests span optimization, machine learning, complex networks, and distributed systems.

He was the first-place winner and a gold medalist at the 21st National Physics Olympiad in Iran. His work has been recognized with several awards, including the 2016 ACM GREENMETRICS Best Student Paper Award, the Amazon Fellowship in Artificial Intelligence, the PIMCO Graduate Fellowship, the Computing and Mathematical Sciences Fellowship, the Annenberg Graduate Fellowship, and the Sharif University of Technology President's Award.

Farshad Lahouti (SM'13) received the B.Sc. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 1997, and the Ph.D. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 2002. In 2005, he joined the School of Electrical and Computer Engineering, University of Tehran, as a Faculty Member, where he founded the Center for Wireless Multimedia Communications. He was the Head of the Communications Engineering Department from 2008 to 2012. He is currently a Visiting Professor of Electrical Engineering with the California Institute of Technology, Pasadena, CA, USA, where he initiated the Digital Ventures Design Program. His current research interests are coding and information theory, signal processing, and communication theory with applications to wireless networks, cyber-physical systems and man–machine symbiosis, and biological and neuronal networks. He received the Distinguished Scientist Award from the Iran National Academy of Sciences in 2014.

Amir Salman Avestimehr (SM'17) received the B.S. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2003, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California at Berkeley, Berkeley, CA, USA, in 2005 and 2008, respectively. He was a Postdoctoral Scholar with the Center for the Mathematics of Information, California Institute of Technology, in 2008. He is currently an Associate Professor with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA. His research interests include information theory, communications, distributed computing, and data analytics. He has received a number of awards, including the Communications Society and Information Theory Society Joint Paper Award, the Presidential Early Career Award for Scientists and Engineers for pushing the frontiers of information theory through its extension to complex wireless information networks, the Young Investigator Program Award from the U.S. Air Force Office of Scientific Research, the National Science Foundation CAREER Award, and the David J. Sakrison Memorial Prize for outstanding research. He was a recipient (as Advisor) of the Qualcomm Innovation Award and the Best Student Paper Award at the IEEE International Symposium on Information Theory. He has been a Guest Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY Special Issue on Interference Networks and General Co-Chairs of the 2012 North America Information Theory Summer School, the 2012 Workshop on Interference Networks, and the 2020 IEEE International Symposium on Information Theory. He is currently an Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY.

Babak Hassibi (M'08) was born in Tehran, Iran, in 1967. He received the B.S. degree from the University of Tehran, Tehran, Iran, in 1989, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, USA, in 1993 and 1996, respectively, all in electrical engineering.

Since January 2001, he has been with the California Institute of Technology, Pasadena, CA, USA, where he is currently the Mose and Lilian S. Bohn Professor of Electrical Engineering. From 2013 to 2016, he was the Gordon M. Binder/Amgen Professor of Electrical Engineering and from 2008 to 2015, he was an Executive Officer of Electrical Engineering, as well as the Associate Director of Information Science and Technology. From October 1996 to October 1998, he was a Research Associate with the Information Systems Laboratory, Stanford University, and from November 1998 to December 2000, he was a Member of the Technical Staff with the Mathematical Sciences Research Center, Bell Laboratories, Murray Hill, NJ, USA. He has also held short-term appointments at Ricoh California Research Center, the Indian Institute of Science, and Linkoping University, Sweden. He is the coauthor of the books (both with A.H. Sayed and T. Kailath) *Indefinite Quadratic Estimation and Control: A Unified Approach to H^2 and H^∞ Theories* (SIAM, 1999) and *Linear Estimation* (Prentice Hall, 2000). His research interests include communications and information theory, control and network science, and signal processing and machine learning. He is a recipient of an Alborz Foundation Fellowship, the 1999 O. Hugo Schuck best paper award of the American Automatic Control Council (with H. Hindi and S.P. Boyd), the 2002 National Science Foundation Career Award, the 2002 Okawa Foundation Research Grant for Information and Telecommunications, the 2003 David and Lucille Packard Fellowship for Science and Engineering, the 2003 Presidential Early Career Award for Scientists and Engineers (PECASE), and the 2009 Al-Marai Award for Innovative Research in Communications, and was a participant in the 2004 National Academy of Engineering "Frontiers in Engineering" program.

He has been a Guest Editor for the IEEE TRANSACTIONS ON INFORMATION THEORY Special Issue on "Space-time transmission, reception, coding and signal processing," was an Associate Editor for Communications of the IEEE TRANSACTIONS ON INFORMATION THEORY during 2004–2006, and is currently an Editor for the journal *Foundations and Trends in Information and Communication*" and for the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING. He is an IEEE Information Theory Society Distinguished Lecturer for 2016–2017.