DISTRIBUTED SOLUTION OF LARGE-SCALE LINEAR SYSTEMS VIA ACCELERATED PROJECTION-BASED CONSENSUS

Navid Azizan-Ruhi[⋆]*, Farshad Lahouti*[⋆]*, Salman Avestimehr*[†]*, Babak Hassibi*[⋆]

[⋆] California Institute of Technology, Pasadena, CA 91125
[†] University of Southern California, Los Angeles, CA 90007

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 (APC)* for this problem. 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 ADMM. 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.

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

1. 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 in a distributed fashion, 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 with the help of a set of computing machines/cores (Figure 1).

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 [1, 2, 3] and its variants (e.g. Nesterov's accelerated gradient [4] and heavy-ball method [5]), as well as the so-called Alternating Direction Method of Multipliers (ADMM) [6] and its variants. ADMM has been widely used [7, 8, 9] for solving various convex optimization problems in a distributed way, and in particular for consensus optimization [10, 11, 12], 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 for solving systems of linear equations. The most famous one of these is what is known as the block Cimmino method [13, 14, 15], which is a block row-projection method [16], and is in a way a distributed implementation of the Kaczmarz method [17]. Another algorithm has been recently proposed in [18, 19], where a consensus-based scheme is used to solve a system of linear equations over a network of autonomous agents.

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

Email: azizan@caltech.edu

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

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, as well as a detailed comparison with all the other distributed methods mentioned above. 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. Finally, as a further implication of our results, we propose a novel distributed preconditioning method, which can be used to improve the convergence rate of distributed gradient-based methods.

2. 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 ($N = n$).

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.

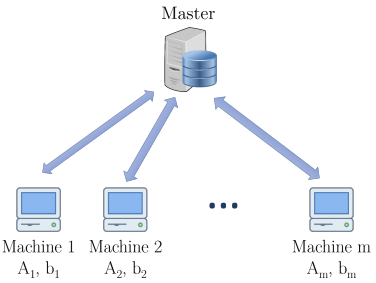

Fig. 1. Schematic representation of the taskmaster and the m machines. Each machine i has only a subset of the equations, i.e. $[A_i, b_i]$.

3. ACCELERATED PROJECTION-BASED CONSENSUS

3.1. 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 $P_i = I - A_i^T(A_iA_i^T)^{-1}A_i$ is the projection matrix onto the nullspace of A_i (It is easy to check that $A_iP_i = 0$ 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. Moreover, it turns out that the block Cimmino method is in fact a special case of APC for $\gamma = 1$.

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 show that

$$\begin{aligned} x_i(t+1) = \underset{x_i}{\text{argmin}} \quad & \|x_i - \bar{x}(t)\|^2 + \frac{1-\gamma}{\gamma}\|x_i - x_i(t)\|^2 \\ \text{s.t.} \quad & A_i x_i = b_i \end{aligned}$$

The second term in the objective is what distinguishes this method from the block Cimmino method. If one sets γ equal

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 $A_i x = b_i$.

for $t = 1$ **to** T **do**

  for each machine i **parallel do**

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

  end for

  at the master: $\bar{x}(t+1) \leftarrow \frac{\eta}{m} \sum_{i=1}^m x_i(t+1) + (1-\eta)\bar{x}(t)$

end for

to 1 (as it is in 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 [18, 19]. 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

$$x_i(t+1) = x_i(t) + \gamma P_i(\bar{x}(t) - x_i(t)), \quad i \in [m], \tag{2a}$$

$$\bar{x}(t+1) = \frac{\eta}{m}\sum_{i=1}^m x_i(t+1) + (1-\eta)\bar{x}(t), \tag{2b}$$

which leads to Algorithm 1.

3.2. Convergence Analysis

We analyze the convergence of the proposed algorithm and prove that it has linear convergence, 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

$$X \triangleq \frac{1}{m}\sum_{i=1}^m A_i^T (A_i A_i^T)^{-1} A_i. \tag{3}$$

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

$$p_i(\lambda;\gamma,\eta) \triangleq \lambda^2 + (-\eta\gamma(1-\mu_i) + \gamma - 1 + \eta - 1)\lambda + (\gamma-1)(\eta-1) \tag{4}$$

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.

$$S = \{(\gamma,\eta) \in [0,2]\times\mathbb{R} \mid \text{roots of } p_i \text{ have magnitude less than 1 for all } i\}. \tag{5}$$

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 $(\gamma,\eta) \in S$. Furthermore, the optimal rate of convergence is*

$$\rho = \frac{\sqrt{\kappa(X)} - 1}{\sqrt{\kappa(X)} + 1} \approx 1 - \frac{2}{\sqrt{\kappa(X)}}, \tag{6}$$

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

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

For proof see the supplementary material in the online version [20].

3.3. Computational Complexity and Numerical Stability

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 (e.g. using QR factorization), which takes another np operations. Thus the overall computational complexity of each iteration is $2pn$.

Finally, we should mention 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.

4. COMPARISON WITH RELATED METHODS

As mentioned earlier, (1) can also be viewed as an optimization problem of the form $\min_x \|Ax-b\|^2$, and since the objective is separable in the data, i.e. $\|Ax-b\|^2 = \sum_{i=1}^m \|A_i x -$

Table 1. A summary of the convergence rates of different methods. The smaller the convergence rate is, the faster is the method. The condition number of X is typically much smaller than that of A^TA.

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

$b_i\|^2$, generic distributed optimization methods such as distributed gradient descent (DGD), distributed Nesterov's accelerated gradient descent (D-NAG), distributed heavy-ball method (D-HBM), Alternating Direction Method of Multipliers (ADMM) apply well to the problem. For a detailed analysis of these methods, see the extended version [20].

The block Cimmino method [13, 14, 15] and the consensus method of [18, 19] are perhaps the closest algorithms in spirit to APC. However, the convergence rate of the Cimmino method is much slower in comparison with APC (its convergence time is the square of that of APC), and in fact APC method includes the block Cimmino method as a special case for $\gamma = 1$. The analysis and proofs can be found in [20]. Although the consensus method of [18, 19] has the advantage that it works for an arbitrary networked architecture, it is extremely slow (convergence rate $= 1 - \mu_{\min}(X)$).

Table 1 shows a summary of the optimal convergence rates of different methods. All the methods have the same per iteration complexity.

5. EXPERIMENTAL RESULTS

We evaluate the performance of the proposed method by comparing it with the other distributed methods discussed throughout the paper. We use randomly-generated problems as well as real-world ones form the National Institute of Standards and Technology repository, *Matrix Market* [21].

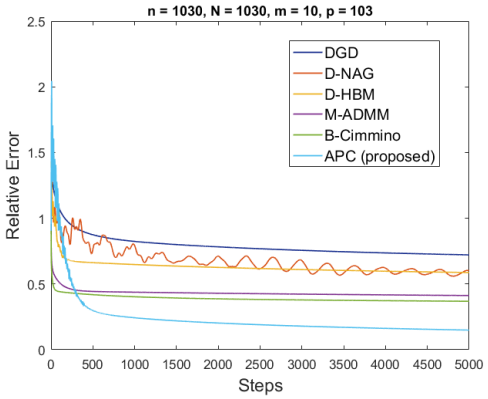

Fig. 2. The decay of the error for different distributed algorithms, on a real problem (ORSIRR 1: Oil reservoir simulation) from Matrix Market [21]. For fairness, the parameters in all the methods have been tuned to their optimal values.

Fig. 2 shows the relative error (the distance from the true solution, divided by the true solution, in ℓ_2 norm) for all the discussed methods, on an example (ORSIRR 1: Oil Reservoir Simulation) from the repository. To make the comparison between different methods fair, we have tuned the parameters in all of the methods to their optimal values. Also, as mentioned before, all the algorithms have the same per-iteration complexity. As one can see, APC outperforms the other methods significantly. For more extensive numerical results see [20].

6. 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 $A_ix = b_i$ by $(A_iA_i^T)^{-1/2}$, which can be done in parallel (locally) with $O(p^2n)$ operations. This transforms the global system of equations $Ax = b$ to a new one $Cx = d$, where

$$C = \begin{bmatrix} (A_1A_1^T)^{-1/2}A_1 \\ \vdots \\ (A_mA_m^T)^{-1/2}A_m \end{bmatrix}, \quad \text{and} \quad d = \begin{bmatrix} (A_1A_1^T)^{-1/2}b_1 \\ \vdots \\ (A_mA_m^T)^{-1/2}b_m \end{bmatrix}.$$

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 $\kappa = \kappa(C^TC) = \kappa(X)$.

7. 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.

8. REFERENCES

[1] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[2] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[3] Kun Yuan, Qing Ling, and Wotao Yin, "On the convergence of decentralized gradient descent," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.

[4] Yurii Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," in *Soviet Mathematics Doklady*, 1983, vol. 27, pp. 372–376.

[5] Boris T Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[6] 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*, vol. 3, no. 1, pp. 1–122, 2011.

[7] Bingsheng He and Xiaoming Yuan, "On the $O(1/n)$ convergence rate of the Douglas-Rachford alternating direction method," *SIAM Journal on Numerical Analysis*, 2012.

[8] Wei Deng and Wotao Yin, "On the global and linear convergence of the generalized alternating direction method of multipliers," *Journal of Scientific Computing*, 2012.

[9] Ruiliang Zhang and James T Kwok, "Asynchronous distributed ADMM for consensus optimization.," in *ICML*, 2014, pp. 1701–1709.

[10] João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Puschel, "D-ADMM: A communication-efficient distributed algorithm for separable optimization," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2718–2723, 2013.

[11] Wei Shi, Qing Ling, Kun Yuan, Gang Wu, and Wotao Yin, "On the linear convergence of the ADMM in decentralized consensus optimization.," *IEEE Trans. Signal Processing*, vol. 62, no. 7, pp. 1750–1761, 2014.

[12] Layla Majzoubi and Farshad Lahouti, "Analysis of distributed ADMM algorithm for consensus optimization in presence of error," in *Proceedings of the 2016 IEEE Confs. Audio, Speech and Signal Processing (ICASSP)*, Mar. 2016.

[13] Iain S Duff, Ronan Guivarch, Daniel Ruiz, and Mohamed Zenadi, "The augmented block Cimmino distributed method," *SIAM Journal on Scientific Computing*, vol. 37, no. 3, pp. A1248–A1269, 2015.

[14] Fridrich Sloboda, "A projection method of the Cimmino type for linear algebraic systems," *Parallel computing*, vol. 17, no. 4-5, pp. 435–442, 1991.

[15] Mario Arioli, Iain Duff, Joseph Noailles, and Daniel Ruiz, "A block projection method for sparse matrices," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 1, pp. 47–70, 1992.

[16] Randall Bramley and Ahmed Sameh, "Row projection methods for large nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 1, pp. 168–193, 1992.

[17] Stefan Kaczmarz, "Angenäherte auflösung von systemen linearer gleichungen," *Bulletin International de lAcademie Polonaise des Sciences et des Lettres*, vol. 35, pp. 355–357, 1937.

[18] Ji Liu, Shaoshuai Mou, and A Stephen Morse, "An asynchronous distributed algorithm for solving a linear algebraic equation," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 2013, pp. 5409–5414.

[19] Shaoshuai Mou, Ji Liu, and A Stephen Morse, "A distributed algorithm for solving a linear algebraic equation," *IEEE Transactions on Automatic Control*, vol. 60, no. 11, pp. 2863–2878, 2015.

[20] Navid Azizan-Ruhi, Farshad Lahouti, Salman Avestimehr, and Babak Hassibi, "Distributed solution of large-scale linear systems via accelerated projection-based consensus," *arXiv preprint arXiv:1708.01413*, 2017.

[21] "Matrix market," `http://math.nist.gov/MatrixMarket/`, Accessed: May 2017.