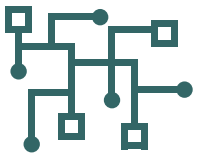

ISE

Industrial and
Systems Engineering

Partitioning Data on Features or Samples in Communication-Efficient Distributed Optimization?

Chenxin Ma[1] and Martin Takáč[1]

[1]Lehigh University

ISE Technical Report 15T-015

LEHIGH
UNIVERSITY

Partitioning Data on Features or Samples in Communication-Efficient Distributed Optimization?

Chenxin Ma
Industrial and Systems Engineering
Lehigh University, USA
chm514@lehigh.edu

Martin Takáč
Industrial and Systems Engineering
Lehigh University, USA
takac.mt@gmail.com

Abstract

In this paper we study the effect of the way that the data is partitioned in distributed optimization. The original DiSCO algorithm [Communication-Efficient Distributed Optimization of Self-Concordant Empirical Loss, Yuchen Zhang and Lin Xiao, 2015] partitions the input data based on samples. We describe how the original algorithm has to be modified to allow partitioning on features and show its efficiency both in theory and also in practice.

1 Introduction

As the size of the datasets becomes larger and larger, distributed optimization methods for machine learning have become increasingly important [2, 5, 13]. Existing mehods often require a large amount of communication between computing nodes [17, 7, 9, 18], which is typically several magnitudes slower than reading data from their own memory [10]. Thus, distributed machine learning suffers from the communication bottleneck on real world applications.

In this paper we focus on the regularized empirical risk minimization problem. Suppose we have n data samples $\{x_i, y_i\}_{i=1}^n$, where each $x_i \in \mathbb{R}^d$ (i.e. we have d features), $y_i \in \mathbb{R}$. We will denote by $X := [x_1, ..., x_n] \in \mathbb{R}^{d\times n}$. The optimization problem is to minimize the regularized empirical loss (ERM)

$$f(w) := \frac{1}{n}\sum_{i=1}^{n} \phi_i(w, x_i) + \frac{\lambda}{2}\|w\|_2^2, \tag{1}$$

where the first part is the *data fitting term*, $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a loss function which typically depends on y_i. Some popular loss functions includes hinge loss $\phi_i(w, x_i) = \max\{0, 1 - y_i w^T x_i\}$, square loss $\phi_i(w, x_i) = (y_i - w^T x_i)^2$ or logistic loss $\phi_i(w, x_i) = \log(1 + \exp(-y_i w^T x_i))$. The second part of objective function (1) is ℓ_2 regularizer ($\lambda > 0$) which helps to prevent over-fitting of the data.

We assume that the loss function ϕ_i is convex and self-concordant [19]:

Assumption 1. *For all $i \in [n] := \{1, 2, \ldots, n\}$ the convex function ϕ is self-concordant with parameter M i.e. the following inequality holds:*

$$|u^T(f'''(w)[u])u| \le M(u^T f''(w)u)^{\frac{3}{2}} \tag{2}$$

for any $u \in \mathbb{R}^d$ and $w \in dom(f)$, where $f'''(w)[u] := \lim_{t\to 0} \frac{1}{t}(f''(w + tu) - f''(w))$.

There has been an enormous interest in large-scale machine learning problems and many parallel [4, 11] or distributed algorithms have been proposed [1, 16, 12, 14, 8].

The main bottleneck in distributed computing –communication– was handled by many researches differently. Some work considered ADMM type methods [3, 6], another used block-coordinate type algorithms [8, 17, 7, 9], where they tried to solve the local sub-problems more accurately (which should decrease the overall communications requirements when compared with more basic approaches [15, 16]).

Algorithm 1 High-level DiSCO algorithm

Input: parameters $\rho, \mu \geq 0$, number of iterations K
Initializing w_0.
for k = 0,1,2,...,K **do**
  Option 1: Given w_k, run DiSCO-S PCG Algorithm 2, get v_k and δ_k
  Option 2: Given w_k, run DiSCO-F PCG Algorithm 3, get v_k and δ_k
  Update $w_{k+1} = w_k - \frac{1}{1+\delta_k} v_k$
end for
Output: w_{K+1}

2 Algorithm

We assume that we have m machines (computing nodes) available which can communicate between each other over the network. We assume that the space needed to store the data matrix X exceeds the memory of every single node. Thus we have to split the data (matrix X) over the m nodes. The natural question is: *How to split the data into m parts?* There are many possible ways, but two obvious ones:

1. split the data matrix X by rows (i.e. create m blocks by rows); Because rows of X corresponds to features, we will denote the algorithm which is using this type of partitioning as ***DiSCO-F***;
2. split the data matrix X by columns; Let us note that columns of X corresponds to samples we will denote the algorithm which is using this type of partitioning as ***DiSCO-S***;

Notice that the DiSCO-S is exactly the same as DiSCO proposed and analyzed in [19]. In each iteration of Algorithm 1, wee need to compute an inexact Newton step v_k such that $\|f''(w_k)v_k - f'(w_k)\|_2 \leq \epsilon_k$, which is an approximate solution to the Newton system $f''(w_k)v_k = \nabla f(w_k)$. The discussion about how to choose ϵ_k and K and a convergence guarantees for Algorithm 1 can be found in [19]. The main goal of this work is to analyze the algorithmic modifications to DiSCO-S when the partitioning type is changed. It will turn out that partitioning on features (DiSCO-F) can lead to algorithm which uses less communications (depending on the relations between d and n) (see Section 3).

DiSCO-S Algorithm. If the dataset is partitioned by samples, such that j–th node will only store $X_j = [x_{j,1}, ..., x_{j,n_j}] \in \mathbb{R}^{d \times n_j}$, which is a part of X, then each machine can evaluate a local empirical loss function

$$f_j(w) := \frac{1}{n_j} \sum_{i=1}^{n_j} \phi(w, x_{j,i}) + \frac{\lambda}{2}\|w\|_2^2. \tag{3}$$

Algorithm 2 Distributed DiSCO-S: PCG algorithm – data partitioned by samples

Input: $w_k \in \mathbb{R}^d$, and $\mu \geq 0$. communication (Broadcast $w_k \in \mathbb{R}^d$ and reduceAll $\nabla f_i(w_k) \in \mathbb{R}^d$)
Initialization: Let P be computed as (4). $v_0 = 0$, $s_0 = P^{-1}r_0$, $r_0 = \nabla f(w_k)$, $u_0 = s_0$.
for $t = 0, 1, 2, ...$ **do**
  Compute Hu_t communication (Broadcast $u_t \in \mathbb{R}^d$ and reduceAll $f''_i(w_k)u_t \in \mathbb{R}^d$)
  Compute $\alpha_t = \frac{\langle r_t, s_t \rangle}{\langle u_t, Hu_t \rangle}$
  Update $v_{(t+1)} = v_t + \alpha_t u_t, Hv_{(t+1)} = Hv_t + \alpha_t Hu_t, r_{t+1} = r_t - \alpha_t Hu_t$.
  Update $s_{(t+1)} = P^{-1}r_{(t+1)}$.
  Compute $\beta_t = \frac{\langle r_{(t+1)}, s_{(t+1)} \rangle}{\langle r_t, s_t \rangle}$
  Update $u_{(t+1)} = s_{(t+1)} + \beta_t u_t$.
  until: $\|r_{(r+1)}\|_2 \leq \epsilon_k$
end for
Return: $v_k = v_{t+1}$, $\delta_k = \sqrt{v_{(t+1)}^T Hv_t + \alpha_t v_{(t+1)}^T Hu_t}$

Because $\{X_j\}$ is a partition of X we have $\sum_{j=1}^{m} n_j = n$, our goal now becomes to minimize the function $f(w) = \frac{1}{m}\sum_{h=1}^{m} f_j(w)$. Let H denote the Hessian $f''(w_k)$. For simplicity in this paper we consider only square loss and hence in this case $f''(w_k)$ is constant (independent on w_k).

In Algorithm 2, each machine will use its local data to compute the local gradient and local Hessian and then aggregate them together. We also have to choose one machine as the master, which computes all the vector operations of PCG loops (Preconditioned Conjugate Gradient), i.e., step 5-9 in Algorithm 2.

The preconditioning matrix for PCG is defined only on master node and consists of the local Hessian approximated by a subset of data available on master node with size τ, i.e.

$$P = \frac{1}{\tau}\sum_{j=1}^{\tau} \phi^{''}(w, x_{1,j}) + \mu I, \tag{4}$$

where μ is a small regularization parameter. Algorithm 2 presents the distributed PCG mathod for solving the preconditioning linear system

$$P^{-1}Hv_k = P^{-1}\nabla f(w_k). \tag{5}$$

DiSCO-F Algorithm. If the dataset is partitioned by features, then jth machine will store $X_j = [a_1^{[j]}, ..., a_n^{[j]}] \in \mathbb{R}^{d_j \times n}$, which contains all the samples, but only with a subset of features. Also, each machine will only store $w_k^{[j]} \in \mathbb{R}^{d_j}$ and thus only be responsible for the computation and updates of $\mathbb{R}^{d_j}$ vectors. By doing so, we only need one ReduceAll on a vector of length n, in addition to two ReduceAll on scalars number.

Algorithm 3 Distributed DiSCO-F: PCG algorithm – data partitioned by features

1: **Input:** $w_k^{[i]} \in \mathbb{R}^{d_i}$ for $i = 1, 2, ..., m$, and $\mu \geq 0$.
2: **Initialization:** Let P be computed as (4). $v_0^{[i]} = 0$, $s_0^{[i]} = (P^{-1})^{[i]} r_0^{[i]}$, $r_0^{[i]} = f'(w_k^{[i]})$, $u_0^{[i]} = s_0^{[i]}$.
3: **while** $\|r_{r+1}\|_2 \leq \epsilon_k$ **do**
4: Compute $(Hu_t)^{[i]}$. communication (ReduceAll an $\mathbb{R}^{d_i}$ vector)
5: Compute $\alpha_t = \frac{\sum_{i=1}^{m}\langle r_t^{[i]}, s_t^{[i]}\rangle}{\sum_{i=1}^{m}\langle u_t^{[i]}, (Hu_t)^{[i]}\rangle}$. communication (ReduceAll a number)
6: Update $v_{t+1}^{[i]} = v_t^{[i]} + \alpha_t u_t^{[i]}$, $(Hv_{t+1})^{[i]} = (Hv_t)^{[i]} + \alpha_t (Hu_t)^{[i]}$, $r_{t+1}^{[i]} = r_t^{[i]} - \alpha_t (Hu_t)^{[i]}$.
7: Update $s_{t+1}^{[i]} = (P^{-1})^{[i]} r_{t+1}^{[i]}$.
8: Compute $\beta_t = \frac{\sum_{i=1}^{m}\langle r_{t+1}^{[i]}, s_{t+1}^{[i]}\rangle}{\sum_{i=1}^{m}\langle r_t^{[i]}, s_t^{[i]}\rangle}$. communication (ReduceAll a number)
9: Update $u_{t+1}^{[i]} = s_{t+1}^{[i]} + \beta_t u_t^{[i]}$.
10: **end while**
11: Compute $\delta_k^{[i]} = \sqrt{{v_{t+1}^{[i]}}^T (Hv_t)^{[i]} + \alpha_t {v_{t+1}^{[i]}}^T (Hu_t)^{[i]}}$.
12: **Integration:** $v_k = [v_{t+1}^{[1]}, ..., v_{t+1}^{[m]}]$, $\delta_k = [\delta_{t+1}^{[1]}, ..., \delta_{t+1}^{[m]}]$ communication (Reduce an $\mathbb{R}^{d_i}$ vector)
13: **Return:** v_k, δ_k

Comparison of Communication and Computational Cost. In Table 1 we compare the communication cost for the two approaches DiSCO-S/DiSCO-F. As it is obvious from the table, DiSCO-F requires only one reduceAll of a vector of length n, whereas the DiSCO-S needs one reduceAll of a vector of length d and one broadcast of vector of size d. So roughly speaking, when $n < d$ then DiSCO-F will need less communication. However, very interestingly, the advantage of DiSCO-F is the fact that it uses CPU on every node more effectively. It also requires less total amount of work to be performed on each node, leading to more balanced and efficient utilization of nodes.

3 Numerical Experiments

We present experiments on several standard large real-world datasets: news20.binary ($d = 1,355,191; n = 19,996; 0.13GB$); kdd2010(test) ($d = 29,890,095; n = 748,401; 0.19GB$); and epsilon ($d = 2,000; n =$

Table 1: Comparison of computation and communication between different ways of partition on data.

			partition by samples	partition by features
computation	master	matrix-vector multiplication	$1(\mathbb{R}^{d\times d}\times\mathbb{R}^{d})$	$1(\mathbb{R}^{d_1\times d_1}\times\mathbb{R}^{d_1})$
		back solving linear system	$1\ (\mathbb{R}^{d})$	$1\ (\mathbb{R}^{d_1})$
		sum of vectors	$4\ (\mathbb{R}^{d})$	$4\ (\mathbb{R}^{d_1})$
		inner product of vectors	$4\ (\mathbb{R}^{d})$	$4\ (\mathbb{R}^{d_1})$
	nodes	matrix-vector multiplication	$1\ (\mathbb{R}^{d\times d}\times\mathbb{R}^{d})$	$1(\mathbb{R}^{d_1\times d_i}\times\mathbb{R}^{d_i})$
		back solving linear system	0	$1\ (\mathbb{R}^{d_i})$
		sum of vectors	0	$4\ (\mathbb{R}^{d_i})$
		inner product of vectors	0	$4\ (\mathbb{R}^{d_i})$
communication	Broadcast		one $\mathbb{R}^{d}$ vector	0
	ReduceAll		one $\mathbb{R}^{d}$ vector	one $\mathbb{R}^{n}$ vector, 2 $\mathbb{R}^{1}$

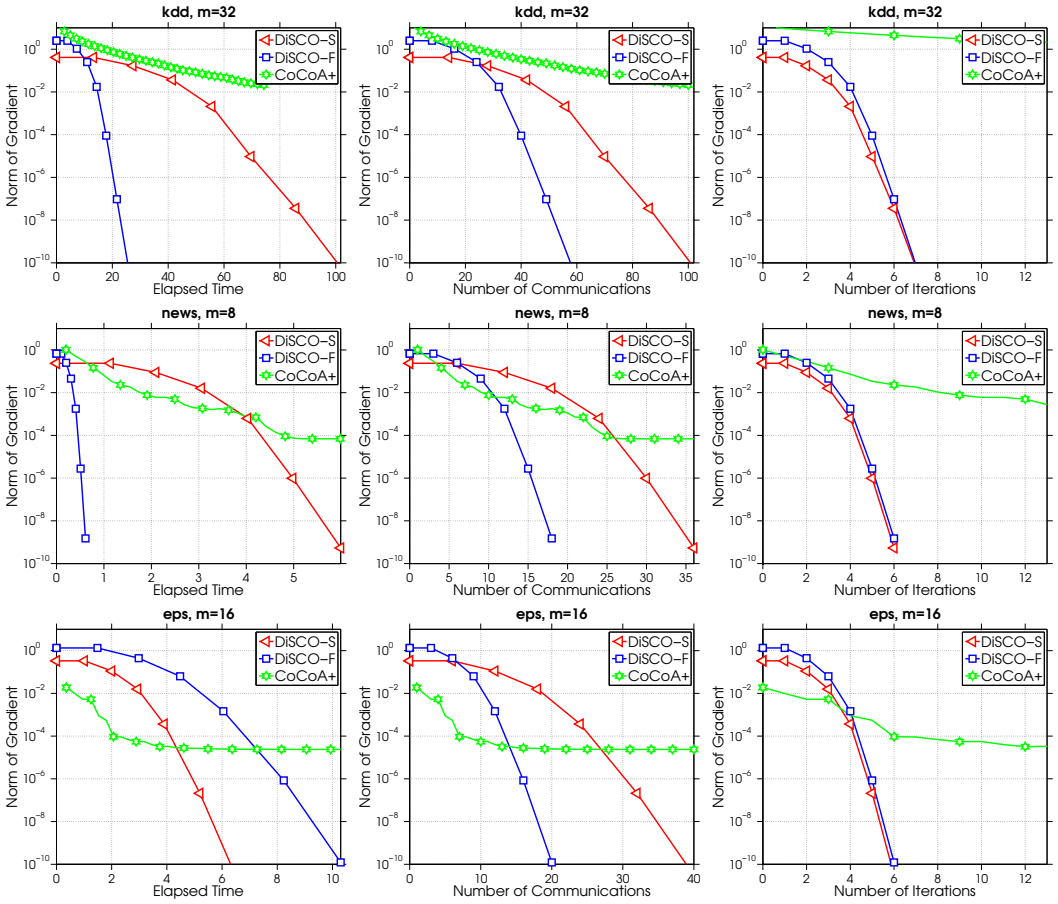

Figure 1: Comparison of DiSCO-S, DiSCO-F and CoCoA+ on various datasets.

$100,000; 3.04GB$). Each data was split into m machines. We implement DiSCO-S, DiSCO-F and CoCoA+ [9] algorithms for comparison in C++, and run them on the Amazon cloud, using 4 m3.xlarge EC2 instances. Figure 1 compares the evolution of $\|\nabla f(w)\|$ as function of elapsed time, number of communications and iterations. As it can be observed, the DiSCO-F needs almost the same number of iterations as DiSCO-S, however, it needs roughly just half the communication, therefore it is much faster (if we care about elapsed time).

References

[1] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

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

[3] 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.

[4] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[5] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13(1):165–202, 2012.

[6] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, pages 1–28, 2012.

[7] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

[8] Ching-Pei Lee and Dan Roth. Distributed box-constrained quadratic optimization for dual linear SVM. ICML, 2015.

[9] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. In *ICML 2015 - Proceedings of the 32th International Conference on Machine Learning*, volume 37, pages 1973–1982. JMLR, 2015.

[10] Jakub Marecek, Peter Richtárik, and Martin Takác. Distributed block coordinate descent for minimizing partially separable functions. *Numerical Analysis and Optimization 2014, Springer Proceedings in Mathematics and Statistics*, 2014.

[11] 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*, pages 693–701, 2011.

[12] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *arXiv preprint arXiv:1310.2059*, 2013.

[13] Ohad Shamir and Nathan Srebro. Distributed stochastic optimization and learning. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pages 850–857. IEEE, 2014.

[14] Ohad Shamir, Nathan Srebro, and Tong Zhang. Communication efficient distributed optimization using an approximate newton-type method. *arXiv preprint arXiv:1312.7853*, 2013.

[15] Martin Takáč, Avleen Bijral, Peter Richtárik, and Nathan Srebro. Mini-batch primal and dual methods for SVMs. *ICML*, 2013.

[16] Martin Takáč, Peter Richtárik, and Nathan Srebro. Distributed mini-batch SDCA. *arXiv preprint arXiv:1507.08322*, 2015.

[17] Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 629–637, 2013.

[18] Tianbao Yang, Shenghuo Zhu, Rong Jin, and Yuanqing Lin. Analysis of distributed stochastic dual coordinate ascent. *arXiv preprint arXiv:1312.1031*, 2013.

[19] Yuchen Zhang and Lin Xiao. Communication-efficient distributed optimization of self-concordant empirical loss. *arXiv preprint arXiv:1501.00263*, 2015.