Data/Feature Distributed Stochastic Coordinate Descent for Logistic Regression

Dongyeop Kang
KAIST Institute
dykang@itc.kaist.ac.kr

Woosang Lim
KAIST
quasar17@kaist.ac.kr

Kijung Shin
Seoul National University
koreaskj@snu.ac.kr

Lee Sael
SUNY Korea & Stony Brook University
sael@sunykorea.ac.kr

U Kang
KAIST
ukang@cs.kaist.ac.kr

ABSTRACT

How can we scale-up logistic regression, or L_1 regularized loss minimization in general, for Terabyte-scale data which do not fit in the memory? How to design the distributed algorithm efficiently? Although there exist two major algorithms for logistic regression, namely Stochastic Gradient Descent (SGD) and Stochastic Coordinate Descent (SCD), they face limitations in distributed environments. Distributed SGD enables data parallelism (i.e., different machines access different part of the input data), but it does not allow feature parallelism (i.e., different machines compute different subsets of the output), and thus the communication cost is high. On the other hand, Distributed SCD allows feature parallelism, but it does not allow data parallelism and thus is not suitable to work in distributed environments.

In this paper we propose DF-DSCD (Data/Feature Distributed Stochastic Coordinate Descent), an efficient distributed algorithm for logistic regression, or L_1 regularized loss minimization in general. DF-DSCD allows both data and feature parallelism. The benefits of DF-DSCD are (a) full utilization of the capabilities provided by modern distributing computing platforms like MapReduce to analyze web-scale data, and (b) independence of each machine in updating parameters with little communication cost. We prove the convergence of DF-DSCD both theoretically, and also show empirical evidence that it is scalable, handles very high-dimensional data with up to 29 millions of features, and converges 2.2× faster than competitors.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms, Design, Experimentation

Keywords

Logistic Regression, L_1 regularized loss minimization, Distributed Computing, Coordinate Descent, MapReduce, Hadoop

1. INTRODUCTION

How can we scale-up logistic regression, or L_1 regularized loss minimization in general, for Terabyte-scale data which do not fit in the memory? How to design the distributed algorithm efficiently? Logistic regression, or L_1 regularized loss minimization in general, is a crucial task with many applications including biological data mining [11], threat classification [7], text processing [9], matrix factorization [6, 25], anomaly detection [19], etc. The major algorithms for learning the parameter for the logistic regression problem are descent based algorithms, including Stochastic Gradient Descent (SGD) and Stochastic Coordinate Descent (SCD). Due to the growing size of the data, there have been expanding interests in developing parallel or distributed version of the SGD and SCD [5, 28]. Ideally, the distributed algorithm for the logistic regression should have two desired parallelism properties in the distributed computing environment. The first property is *data* or *input parallelism*, which we define as follows.

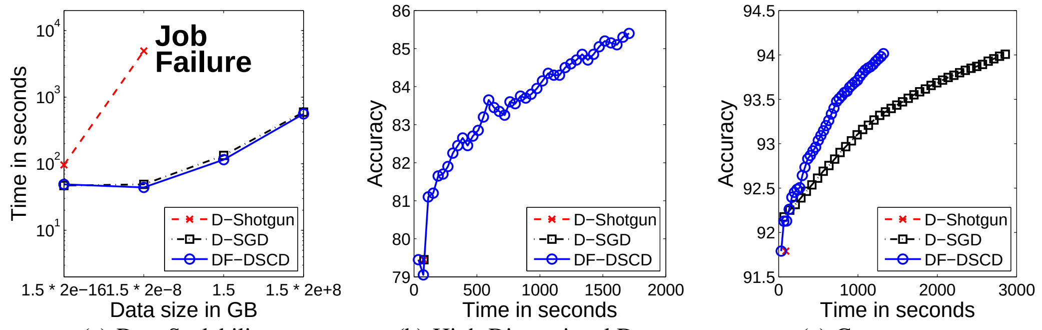
Definition 1 (Data Parallelism). *Data parallelism is the property of distributed algorithm that different machines access different parts of the input data, and the processing of the input data in a machine is not affected by those of other machines.*

Algorithms satisfying data parallelism are run in distributed systems efficiently, since the data can be divided into machines, and each machine focuses only on the piece of the whole data it receives. The second property is *feature* or *output parallelism* which we define as follows.

Definition 2 (Feature Parallelism). *Feature parallelism is the property of distributed algorithm that different machines compute different subsets of the output features, and the computation of the output in a machine is not affected by those of other machines.*

Algorithms satisfying feature parallelism can handle high dimensional features efficiently since the task of updating the features is distributed over machines. Also, the communication cost for exchanging the features becomes much smaller since there is no need to aggregate the features from machines to finalize the features.

Previous works on parallel or distributed version of SGD and SCD satisfy only one of the two desired parallelism properties. Zinkevich et al. [28] developed Parallel SGD (PSGD). The PSGD algorithm provides data parallelism, but it does not provide feature parallelism. Therefore, it is impossible for each machine to select a subset of the output features and update it independently. In the SCD side, Bradley et al. [5] developed ShotGun, a Parallel



(a) Data Scalability (b) High-Dimensional Data Processing (c) Convergence

Figure 1: Performance of proposed DF-DSCD. (a) DF-DSCD outperforms D-Shotgun: DF-DSCD successfully analyzes the synthetic **S1+4** data spanning 207 GB while D-Shotgun fails. (b) DF-DSCD outperforms D-SGD: DF-DSCD runs on 1.3 million-dimensional New20 data while D-SGD fails to run on it. (c) DF-DSCD converges 2.2× faster than D-SGD on RCV1 data: D-SGD requires 2.2× more time than DF-DSCD to achieve the same accuracy. Note that in (b,c), D-Shotgun failed running after the first iteration due to the network overhead.

Table 1: Comparison of methods. Our proposed DF-DSCD is the only one that 1) handles very large data, 2) process high-dimensional data, and 3) provides fast convergence.

	DF-DSCD	**D-SGD**	**D-Shotgun**
Scalable to Large Data	✓	✓	
High Dim. Data	✓		
Fast Convergence	✓		

Stochastic Coordinate Descent (PSCD) algorithm in a shared memory setting. ShotGun provides feature parallelism since each core works independently on subsets of the features. However, it does not provide data parallelism; i.e., each core should read *all* the data.

In this paper, we tackle the following problem: *how can we design a distributed logistic regression, or L_1 regularized loss minimization, algorithm that provides both data and feature parallelism?* Our solution is DF-DSCD, a distributed algorithm that achieves both data and feature parallelism, overcoming the limitations PSGD and PSCD. DF-DSCD provides data parallelism by letting each machine work on different parts of the data independently; furthermore, DF-DSCD provides feature parallelism by letting each machine work on different features. We give theoretical analysis about the convergence property of DF-DSCD. We also show extensive experimental results showing that DF-DSCD is scalable, handles very high dimensional data, and converges faster (up to 2.2×) than its competitors. Table 1 shows the advantage of DF-DSCD compared to other competitors. The main contributions are the followings.

Table 2: Table of symbols.

Symbol	**Definition**
n	number of data instances
d	number of features
$\mathbf{x}_i$	ith d-dimensional predictor variable instance
$\mathbf{X}$	n by d design matrix whose ith row is $\mathbf{x}_i$
y_i	ith response variable instance
$\boldsymbol{\theta}$	d-dimensional weight parameters
${}_i\theta_j^{(t)}$	jth element of $\boldsymbol{\theta}$ updated from ith machine at time t
η	learning rate
ρ	spectral radius of $\mathbf{X}^T\mathbf{X}$
M	number of machines
T	number of iterations
P	number of features updated by each machine in DF-DSCD
C	number of features updated at each iteration in (D-)Shotgun
s	average proportion of nonzero feature values per example

- **Design.** We carefully design DF-DSCD, a data/feature distributed stochastic coordinate descent algorithm for large scale logistic regression. Unlike Distributed Stochastic Gradient Descent (D-SGD) and Distributed Shotgun (D-Shotgun) which provide *either* data or feature parallelism, resp., in distributed environment, DF-DSCD provides *both* data and feature parallelism.
- **Scalability.** DF-DSCD is highly scalable on the data size: DF-DSCD successfully analyzes 207 GB data while D-Shotgun fails to analyze it, as shown in Figure 1(a). Furthermore, DF-DSCD processes very high-dimensional data effectively: DF-DSCD runs on the data with more than millions of features, while D-SGD and D-Shotgun fail to run on it as shown in Figure 1(b).
- **Convergence.** We analyze the convergence properties of DF-DSCD both theoretically and empirically. Theoretically, we prove that DF-DSCD decreases the loss function of the logistic regression at every iteration. Empirically, we present results that DF-DSCD converges up to 2.2× faster than its competitors, as shown in Figure 1(c).

The rest of this paper is organized as follows. Section 2 presents the preliminaries of logistic regression. Section 3 describes our proposed DF-DSCD algorithm. Sections 4 analyzes the convergence properties and the time complexity of DF-DSCD. Section 5 presents the experimental results. After reviewing related works in Section 6, we conclude in Section 7. Table 2 lists the symbols used in this paper.

2. PRELIMINARIES

In this section, we describe the preliminaries of logistic regression and its algorithms.

2.1 Logistic Regression

Logistic regression is a special case of the general problem called the L_1 regularized loss minimization. Let $S = \{(\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R})\}_{i=1}^n$ be a set of n training examples where $\{\mathbf{x}_i \in \mathbb{R}^d\}$ are d-dimensional inputs, and $\{y_i \in \mathbb{R}\}$ are target labels. Let $\mathbf{X}$ be an n by d design matrix whose ith row is $\mathbf{x}_i$. In the logistic regression, the target variable y takes discrete values; for the sake of simplicity we focus on binary classification where y takes either -1 or 1; multi-class generalization is straightforward, too. The probability that y takes the value 1 is given by $p(y = 1|\mathbf{x};\boldsymbol{\theta}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T\mathbf{x}}}$, where $\boldsymbol{\theta} \in \mathbb{R}^d$ are the weight parameters. $p(y = -1|\mathbf{x};\boldsymbol{\theta})$ is defined naturally: $p(y =$

$-1|\mathbf{x};\boldsymbol{\theta}) = 1 - p(y = 1|\mathbf{x};\boldsymbol{\theta})$. The logistic regression problem is to find $\boldsymbol{\theta}$ that minimizes the following loss function based on the negative log likelihood:

$$F(\boldsymbol{\theta}) = \sum_{i=1}^{n} - \log p(y_i|\mathbf{x}_i;\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_1 = \sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{x}_i^T\boldsymbol{\theta})) + \lambda||\boldsymbol{\theta}||_1. \quad (1)$$

where λ is a regularization term to penalize large weight parameters.

2.2 Algorithms for Logistic Regression

We review algorithms for learning the parameter $\boldsymbol{\theta}$ in the logistic regression. The goal is to minimize the loss function $F(\boldsymbol{\theta})$. There are two lines of learning algorithms for the logistic regression: Newton-Rhapson method and descent based methods. The Newton-Rhapson's method is not used practically since the matrix inversion it involves is very expensive, and many of the design matrices are not invertible. For the reason, we focus on the descent based methods which can handle large data.

2.2.1 Gradient Descent

The Gradient Descent (GD) algorithm starts with an initial guess $\boldsymbol{\theta}^{(0)}$ of the parameter $\boldsymbol{\theta}$, and iteratively updates $\boldsymbol{\theta}$ by moving it toward the direction of negative gradient. Since the loss function $F(\boldsymbol{\theta})$ of the logistic regression is convex [12], the GD algorithm always converges to the global minimum. It can be shown that the partial derivative of $F(\boldsymbol{\theta})$ with regard to θ_j, after reformulation with duplicate features as explained in Section 3.3, is given by

$$(\nabla F(\boldsymbol{\theta}))_j := \frac{\partial}{\partial\theta_j}F(\boldsymbol{\theta}) = [\sum_{i=1}^{n}(y_iX_{ij}(p_i - 1))] + \lambda, \quad (2)$$

where $p_i = 1/(1 + exp(-y_i\mathbf{x}_i^T\boldsymbol{\theta}))$. Thus, the GD algorithm is given by iteratively updating $\boldsymbol{\theta}$ using the equation $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla F(\boldsymbol{\theta})$, where η is the positive learning rate. Notice that the parameter $\boldsymbol{\theta}$ is updated incrementally over instances.

Stochastic Gradient Descent (SGD) randomizes iterating through the instances. SGD gets a random approximation of the partial derivatives in much less than O(nd) time where n is the number of instances and d is the number of features, and thus the parameters are updated much more rapidly than in GD. Moreover, if jth feature of a training example $\mathbf{x}$ is 0, then updating θ_j based on $\mathbf{x}$ can be skipped. This means that the time for each iteration is O(nsd) where s is the density (average proportion of nonzero feature values per example). Therefore, for sparse data, SGD is very efficient in practice.

2.2.2 Coordinate Descent

Coordinate Descent (CD) also optimizes for the set of parameters. However, instead of optimizing all parameters at a time, CD updates a single parameter (or a coordinate) θ_j at a time using all the data. That is, $\theta_j \leftarrow \theta_j - \eta(\nabla F(\boldsymbol{\theta}))_j$.

Often, CD converges too slowly to be useful. However, CD can be useful in problems where computing solutions over all of the features is difficult, but computing solutions over a subset of features is relatively easy. Therefore, between the non-parallel version of these two variants (SGD and CD), SGD is the faster algorithm. Stochastic Coordinate Descent (SCD) is a variant of CD where the coordinate to be updated is randomly chosen at each iteration.

2.2.3 Parallel Stochastic Gradient Descent

Mann et al. [15] proposed Parallel Stochastic Gradient Descent (PSGD) by partitioning the data of size n into $\frac{n}{R}$ pieces where R is the number of parallel processors. Further, Zinkevich et al. [28] showed detailed analysis and experimental evidence. In the work, each processor independently learns the parameters of the optimization problem using SGD for multiple iterations. Finally after the last iteration by all processors, the parameters from all processors are collected and averaged to give a final single set of parameters. This method achieves the scalability by dividing the large data into smaller manageable size for each processor. The advantage of only combining the parameters at the end of all iterations reduces the communication overhead of exchanging parameters. The MapReduce version of Parallel Stochastic Gradient Descent is proposed in Section 3.

2.2.4 Parallel Stochastic Coordinate Descent

Bradley et al. [5] proposed ShotGun, a Parallel Stochastic Coordinate Descent (PSCD) algorithm which updates parameters in parallel. Assume that C processors exist in a multi-processor machine. At each iteration, Shotgun randomly selects C coordinates to update, and assigns each coordinate to a distinct processor. Each processor then solves for the assigned parameter using coordinate descent. Each time a parameter changes, it is written to a data structure that is shared by all other processors. Given that reads will not result in any data inconsistency and each processor only writes to parameters that they are responsible for, the data remain consistent. Algorithm 1 shows the Shotgun algorithm [5] which uses the duplicate feature notation described in Section 3.3. In line 6, the $\beta = \frac{1}{4}$ is a constant, and the $(\nabla F(\boldsymbol{\theta}))_j$ is as defined in Eq. (2).

Algorithm 1: Shotgun: Parallel SCD

```
Input  : Set $S = \{(\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0,1\})\}_{i=1}^n$ of $n$ training
         examples,
         Number $C$ of parameters to update per iteration.
Output: $\boldsymbol{\theta}$.
1 $\boldsymbol{\theta} \leftarrow 0$ ;
2 while not converged do
3     Choose random subset of $C$ weights in $\{1, ..., 2d\}$ ;
4     In parallel on $C$ processors
5         // In each processor, update the assigned coordinate
          using $S$ ;
6         Get assigned weight $j$ ;
7         $\boldsymbol{\theta}_j \leftarrow \boldsymbol{\theta}_j + max\{-\boldsymbol{\theta}_j, -(\nabla F(\boldsymbol{\theta}))_j/\beta\}$ ;
8     end
9 end
```

Shotgun achieves scalability by dividing the high dimensional feature space into blocks of features. The MapReduce version of Parallel Stochastic Coordinate Descent is proposed in Section 3.

Comparing PSGD and PSCD, PSGD scales with the number of instances while PSCD scales with the number of features. To take the advantages of both approaches, we propose DF-DSCD algorithm in Section 3.

3. PROPOSED METHOD

In this section, we describe DF-DSCD, our proposed algorithm for fast and scalable logistic regression in distributed environment. We first propose two preliminary algorithms: Distributed Stochastic Gradient Descent (D-SGD) and Distributed Shotgun (D-Shotgun). D-SGD is a MapReduce version of Parallel Stochastic Gradient Descent [28] in distributed environment. D-Shotgun is a MapReduce version of Parallel Coordinate Descent [5] in distributed environment. Unfortunately, each of them has a drawback: D-SGD does not provide feature parallelism, and D-Shotgun does not provide data parallelism. To overcome the problem, in Section 3.3 we propose DF-DSCD which couples the best of both D-SGD and D-Shotgun to provide *both* data and feature parallelism.

3.1 Distributed SGD

Distributed Stochastic Gradient Descent (D-SGD) is a data parallel version of the Stochastic Gradient Descent algorithm. The MapReduce algorithm of D-SGD is shown in Algorithm 2.

Algorithm 2: MapReduce algorithm for D-SGD

```
Input : Set $S = \{(\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0,1\})\}_{i=1}^n$ of $n$ training
        examples,
        Weight $\theta^{(t)} \in \mathbb{R}^d$ at time $t$,
        Learning rate $\eta$.
Output: Weight $\theta^{(t+1)} \in \mathbb{R}^d$ at time $t+1$.
1  D-SGD-Map1(Key k, Value v)
2  begin
3      $(\mathbf{x}_i, y_i) \leftarrow (k, v)$ ;
4      $p \leftarrow$ rand() % number_reducer ;
5      Output($p$, $(\mathbf{x}_i, y_i)$) ;
6  end
7  D-SGD-Reduce1(Key k, Value v[1..r])
8  begin
9      ${}_k\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)}$;
10     foreach $(\mathbf{x}_i, y_i) \in v[1..r]$ do
11         ${}_k\boldsymbol{\theta}^{(t+1)} \leftarrow_k \boldsymbol{\theta}^{(t+1)} - \eta\nabla F({}_k\boldsymbol{\theta}^{(t+1)})$ ;
12     end
13     Output($k$, ${}_k\theta^{(t+1)}$) ;
14 end
```

Given the set $S = \{(\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0,1\})\}_{i=1}^n$ of n training examples, each machine receives an equal number $\frac{|S|}{M}$ of instances in D-SGD where M is the number of machines. This is done by assigning data in mappers. Then, the ith reducer gets S_i which is the ith piece of S after dividing it into M equal pieces. The parameter update is performed in lines 9-12, which is exactly the standard Stochastic Gradient Descent (SGD). The mapper D-SGD-Map1() and the reducer D-SGD-Reduce1() are run for total T iterations.

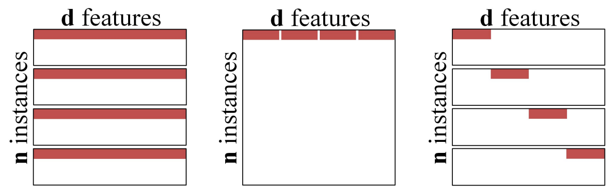
After one iteration of D-SGD-Map1() and D-SGD-Reduce1() procedure, the kth reducer outputs weight ${}_k\boldsymbol{\theta}^{(t+1)}$ where $k \in [1..M]$. The final weight $\boldsymbol{\theta}^{(t+1)}$ is calculated by averaging all ${}_k\boldsymbol{\theta}^{(t+1)}$ from reducers, because each weight is updated from an independent set of training instances (data parallelism). That is, $\boldsymbol{\theta}^{(t+1)} \leftarrow \frac{1}{M}\sum_{k=1}^{M} {}_k\boldsymbol{\theta}^{(t+1)}$.

The weight parameters in D-SGD are passed to the reducers by the distributed cache functionality of MapReduce. D-SGD is illustrated in Figure 2(a), where each black box denotes a piece of data assigned to each machine, and the red rectangles are the features updated from each machine.

D-SGD achieves data parallelism by dividing the data into pieces and assigning each piece into each machine. Each machine can independently work on its own data. However, D-SGD does not achieve feature parallelism: i.e., each machine is not allowed to select a subset of features to update.

3.2 Distributed Shotgun

Distributed Shotgun (D-Shotgun) is a distributed version of the Shotgun [5] (Parallel SCD) algorithm. In D-Shotgun, each machine receives all the data instances, and then updates the parameters assigned to it independently from each other. Of course, this incurs



(a) D-SGD (b) D-Shotgun (c) DF-DSCD

Figure 2: Illustration of D-SGD, D-Shotgun and DF-DSCD. Each black box denotes a piece of data assigned to each machine, and the red rectangles are the features updated from each machine. While D-SGD enables *only* data parallelism by partitioning data instances into machines, and D-Shotgun enables *only* feature parallelism by computing a subset of the output (d) from each machine, DF-DSCD enables *both* data and feature parallelism by partitioning data instances to machines and computing a subset of the output from each machine.

heavy network traffic since the data instances should be broadcast to all the machines. We will see how to overcome this problem in Section 3.3.

The MapReduce algorithm of D-Shotgun is as follows. The main program selects C coordinates to update, and write each of the coordinates in a separate command file; thus, total C command files are created. Each of the file is read by a mapper, and the mapper reads all the data instance in HDFS to perform coordinate descent for the assigned coordinates. Since mappers perform all the computation needed for the coordinate descent, no reducer is needed. D-Shotgun is illustrated in Figure 2(b), where the large black box is all the data, and the red rectangles are outputs from each machine.

D-Shotgun achieves the feature parallelism: i.e., each machine randomly selects a subset of features, and updates them. As demonstrated in Bradley et al. [5] theoretically and empirically, D-Shotgun has strong convergence bound and linear speedup. The upper bound of number of parallel updates is $\frac{d}{2\rho}$ where d is size of feature dimension and ρ is the spectral radius of $\mathbf{X}^T\mathbf{X}$ [5]. Thus, by choosing the optimal number of coordinate to update w.r.t. the given data, D-Shotgun can be effectively optimized. However, D-Shotgun has a significant problem to be used in a distributed environment: it does not provide data parallelism, and thus each machine is forced to access all the data which can be a significant bottleneck, as we will see experimentally in Section 5.

3.3 DF-DSCD: Data/Feature Distributed Stochastic Coordinate Descent

Even though D-SGD and D-Shotgun achieve respectively data or feature parallelism, none of them achieves both. Then, a natural question arises: can we design an algorithm that has both data and feature parallelism? In this section, we propose Data/Feature Distributed Stochastic Coordinate Descent (DF-DSCD), a distributed algorithm for logistic regression which achieves the goal. DF-DSCD takes the best properties of D-SGD and D-Shotgun. As in D-SGD, DF-DSCD let each machine receive the equal number $\frac{|S|}{M}$ of instances where S is the set of training data and M is the number of machines. In other words, the ith machine gets S_i which is the ith piece of S after dividing it into M equal pieces. Also, as in D-Shotgun, DF-DSCD let total PM coordinates be equally assigned to M machines, so that each machine gets P coordinates.

Before describing DF-DSCD algorithm in detail, we transform the original problem into an equivalent problem to ease the description and analysis [5,23]. We let $\hat{\boldsymbol{\theta}} \in \mathbb{R}_+^{2d}$, and use duplicated features $\hat{\mathbf{x}}_i = [\mathbf{x}_i; -\mathbf{x}_i] \in \mathbb{R}^{2d}$. The design matrix turns into an n by $2d$ matrix

Algorithm 3: DF-DSCD: Data/feature distributed stochastic coordinate descent.

```
Input  : Set $S = \{(\hat{\mathbf{x}}_i \in \mathbb{R}^{2d}, y_i \in \{0,1\})\}_{i=1}^n$ of $n$ training
         examples,
         Number $P$ of coordinates to update per machine.
Output: $\hat{\boldsymbol{\theta}}$.
1  $\hat{\boldsymbol{\theta}} \leftarrow 0$ ;
2  while not converged do
3      Choose $PM$ unique coordinates randomly, and divide it
       into equal sized sets $P_1, ..., P_M$ ;
4      Split $S$ into equal sized sets $S_1, ..., S_M$ ;
5      For each machine $M_i$, assign $P_i$ and $S_i$ ;
6      In $M$ distributed machines
7          // In machine $M_k$, update coordinates in $P_k$ using $S_k$;
8          foreach $j \in P_k$ do
9              Set $\eta$ to satisfy Eq. (6) in Section 4;
10             $\hat{\theta}_j \leftarrow \hat{\theta}_j + \eta \cdot max\{-\hat{\theta}_j, -(\nabla F_k(\hat{\boldsymbol{\theta}}))_j/\beta\}$ ;
11         end
12     end
13 end
```

$\hat{\mathbf{X}}$ whose ith row is $\hat{\mathbf{x}}_i$. Then the objective function becomes

$$F(\hat{\boldsymbol{\theta}}) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \hat{\mathbf{x}}_i^T \hat{\boldsymbol{\theta}})) + \lambda \|\hat{\boldsymbol{\theta}}\|_1 \quad (3)$$

If $\hat{\boldsymbol{\theta}} \in \mathbb{R}_+^{2d}$ minimizes Eq. (3), then $\boldsymbol{\theta} \in \mathbb{R}^d$, where $\theta_i = \hat{\theta}_i - \hat{\theta}_{d+i}$, minimizes Eq. (1) [23]. Thus, from this point we aim to find $\hat{\boldsymbol{\theta}}$ which minimizes the loss function in Eq. (3).

Let $\hat{\mathbf{X}}_k$ and y_k be the set of $\hat{\mathbf{x}}$ and y in S_k, respectively. Without loss of generality, we can express $\hat{\mathbf{X}}$ after some permutation as follows:

$$\hat{\mathbf{X}} \in \mathbb{R}^{n \times 2d} = \begin{pmatrix} \hat{\mathbf{X}}_1 \\ \hat{\mathbf{X}}_2 \\ \vdots \\ \hat{\mathbf{X}}_M \end{pmatrix}, \quad \hat{\mathbf{X}}^T = \begin{pmatrix} \hat{\mathbf{X}}_1^T & \hat{\mathbf{X}}_2^T & \cdots & \hat{\mathbf{X}}_M^T \end{pmatrix} \quad (4)$$

We next define the partial loss function $F_k(\hat{\boldsymbol{\theta}})$ computed from the data in S_k:

$$F_k(\hat{\boldsymbol{\theta}}) = \sum_{i \in \hat{\mathbf{X}}_k} \log(1 + \exp(-y_i \hat{\mathbf{x}}_i^T \hat{\boldsymbol{\theta}})) + \frac{\lambda}{M} \|\hat{\boldsymbol{\theta}}\|_1, \quad (5)$$

where we use the notation $i \in \hat{\mathbf{X}}_k$ to specify the row indices of $\hat{\mathbf{X}}$ belonging to $\hat{\mathbf{X}}_k$. Then, $F(\hat{\boldsymbol{\theta}})$ is given by

$$F(\hat{\boldsymbol{\theta}}) = \sum_{k=1}^{M} F_k(\hat{\boldsymbol{\theta}}).$$

The DF-DSCD algorithm is shown in Algorithm 3. The coordinates and data are assigned to machines in lines 3-5. Then, each machine performs parallel SCD on the assigned data where the update equation (line 10) is slightly changed from the standard update equation (line 7 of Algorithm 1). As we will see in Section 4, the objective function $F()$ decreases over iterations in DF-DSCD.

The MapReduce algorithm of DF-DSCD is shown in Algorithm 4. The data assignment is performed in the mapper. The reducer updates the assigned coordinates using the same equation expressed in line 10 of Algorithm 3. Since each reducer updates different coordinates, there is no need to sum up the updated parameters from different reducers. DF-DSCD is illustrated in Figure 2(c), where each black shaded box is a piece of data assigned to each machine, and the red rectangles are outputs from each machine.

Algorithm 4: MapReduce algorithm for DF-DSCD

```
Input  : Set $S = \{(\hat{\mathbf{x}}_i \in \mathbb{R}^{2d}, y_i \in \{0,1\})\}_{i=1}^n$ of $n$ training
         examples,
         Weight $\hat{\boldsymbol{\theta}}^{(t)} \in \mathbb{R}^{2d}$ at time $t$,
         Coordinate sets $P_1, ..., P_M$ with $|P_i| = P$.
Output: Weight $\hat{\boldsymbol{\theta}}^{(t+1)} \in \mathbb{R}^{2d}$ at time $t+1$.
1  DF-DSCD-Map1(Key k, Value v);
2  begin
3      $(\hat{\mathbf{x}}_i, y_i) \leftarrow (k, v)$ ;
4      $p \leftarrow$ rand() % number_reducer ;
5      Output($p$, $(\hat{\mathbf{x}}_i, y_i)$) ;
6  end
7  DF-DSCD-Reduce1(Key k, Value v[1..r]);
8  begin
9      foreach $j \in P_k$ do
10         ${}_k\hat{\theta}_j^{(t+1)} \leftarrow \hat{\theta}_j^{(t)}$ ;
11     end
12     foreach $j \in P_k$ do
13         Set $\eta$ to satisfy Eq. (6) in Section 4;
14         ${}_k\hat{\theta}_j^{(t+1)} \leftarrow {}_k\hat{\theta}_j^{(t+1)} + \eta \cdot max\{-\hat{\theta}_j, -(\nabla F_k(\hat{\boldsymbol{\theta}}))_j/\beta\}$;
15     end
16     foreach $j \in P_k$ do
17         Output($j$, ${}_k\hat{\theta}_j^{(t+1)}$) ;
18     end
19 end
```

DF-DSCD achieves the data parallelism by dividing the data into pieces and assigning each piece into each machine. Each machine can independently work on its own data. Moreover, DF-DSCD achieves the feature parallelism since each machine is allowed to select a subset of features to update.

4. ANALYSIS

In this section, we give a theoretical convergence analysis and time complexity of DF-DSCD.

4.1 Convergence Analysis

We prove that DF-DSCD decreases the loss function $F()$ of the logistic regression problem at each iteration. Since DF-DSCD randomly chooses coordinates, it is necessary to bound the *expectation* of the loss function where the expectation is over the random choices of the coordinates. Our main result is Theorem 1 which states that the expectation of the loss function decreases at each iteration when DF-DSCD is run with a proper small step size. We first present several lemmas, and use them to prove Theorem 1.

Without loss of generality, we assume that $diag(\hat{\mathbf{X}}^T\hat{\mathbf{X}}) = \mathbf{1}$, following [5]. The Hessian of $F(\hat{\boldsymbol{\theta}})$ is given by

$$\frac{\partial^2 F(\hat{\boldsymbol{\theta}})}{\partial \hat{\theta}_j \hat{\theta}_k} = \sum_{i=1}^{n} \hat{X}_{ij} \hat{X}_{ik} (1 - \hat{p}_i) \hat{p}_i,$$

where $\hat{p}_i = 1/(1 + ext(-y_i \hat{\mathbf{x}}_i^T \hat{\boldsymbol{\theta}}))$. Let $\Delta\hat{\boldsymbol{\theta}}$ be the change of $\hat{\boldsymbol{\theta}}$ at each iteration, and $\Delta_{\hat{\theta}_j}^k$ be the jth coordinate of $\Delta\hat{\boldsymbol{\theta}}$ updated from machine k. We first show the upper bound of $F(\hat{\boldsymbol{\theta}} + \Delta\hat{\boldsymbol{\theta}}) - F(\hat{\boldsymbol{\theta}})$.

Lemma 1. *For any $\hat{\mathbf{X}}$, $F(\hat{\boldsymbol{\theta}} + \Delta\hat{\boldsymbol{\theta}}) - F(\hat{\boldsymbol{\theta}}) \le (\Delta\hat{\boldsymbol{\theta}})^T \nabla F(\hat{\boldsymbol{\theta}}) + \frac{\beta}{2} (\Delta\hat{\boldsymbol{\theta}})^T \hat{\mathbf{X}}^T \hat{\mathbf{X}} \Delta\hat{\boldsymbol{\theta}}$, where $\beta = \frac{1}{4}$ is a constant.*

Proof. See the supplementary material [1]. □

Next, we give the relation between $\nabla F(\hat{\boldsymbol{\theta}})$ and $\nabla F_k(\hat{\boldsymbol{\theta}})$.

Lemma 2. $\nabla F(\hat{\boldsymbol{\theta}}) = \sum_{k=1}^{M} \nabla F_k(\hat{\boldsymbol{\theta}})$.

Proof. See the supplementary material [1]. □

Next, we give a loose bound of the difference between $E[F(\hat{\boldsymbol{\theta}} + \Delta\hat{\boldsymbol{\theta}})]$ and $E[F(\hat{\boldsymbol{\theta}})]$.

Lemma 3. *For $M \geq 2$, $E[F(\hat{\boldsymbol{\theta}} + \Delta\hat{\boldsymbol{\theta}}) - F(\hat{\boldsymbol{\theta}})]$ is bounded by*

$$E[F(\hat{\boldsymbol{\theta}} + \Delta\hat{\boldsymbol{\theta}}) - F(\hat{\boldsymbol{\theta}})] \leq PE_j[\sum_{k=1}^{M} \Delta^k_{\hat{\theta}_j} \nabla F(\hat{\boldsymbol{\theta}})_j + \frac{\beta(1+\epsilon)}{2} \sum_{k=1}^{M} (\Delta^k_{\hat{\theta}_j})^2],$$

where $\epsilon = \frac{(PM-1)(\rho-1)}{2d-1}$ and ρ is the spectral radius of $\hat{\mathbf{X}}^T\hat{\mathbf{X}}$.

Proof. See the supplementary material [1]. □

Let $\mathbf{P}(\hat{\boldsymbol{\theta}})$ be a diagonal matrix with $\mathbf{P}(\hat{\boldsymbol{\theta}})_{i,i} = \hat{p}_i$. And let $\mathbf{P}_k(\hat{\boldsymbol{\theta}})$ be a sub-matrix of $\mathbf{P}(\hat{\boldsymbol{\theta}})$ corresponding to $\hat{\mathbf{X}}_k$ and $\mathbf{y}_k$ is a sub-vector of $\mathbf{y}$ corresponding to $\hat{\mathbf{X}}_k$. The gradients of $F(\hat{\boldsymbol{\theta}})$ and $F_k(\hat{\boldsymbol{\theta}})$ are expressed by $\mathbf{P}(\hat{\boldsymbol{\theta}})$ and $\mathbf{P}_k(\hat{\boldsymbol{\theta}})$ as follows:

$$\begin{aligned} \frac{\partial F(\hat{\boldsymbol{\theta}})}{\partial \hat{\theta}_j} &= (\hat{\mathbf{X}}^T(\mathbf{P}(\hat{\boldsymbol{\theta}}) - \mathbf{I})\mathbf{y} + \lambda\mathbf{1})_j \\ \frac{\partial F_k(\hat{\boldsymbol{\theta}})}{\partial \hat{\theta}_j} &= (\hat{\mathbf{X}}_k^T(\mathbf{P}_k(\hat{\boldsymbol{\theta}}) - \mathbf{I}_k)\mathbf{y}_k + \frac{\lambda}{M}\mathbf{1})_j \\ &= \left((\hat{\mathbf{X}}_k^T)_j(\mathbf{P}_k(\hat{\boldsymbol{\theta}}) - \mathbf{I}_k)\mathbf{y}_k + \frac{\lambda}{M}\right) \end{aligned}$$

We give an upper bound of $\sum_{k=1}^{\mathbf{M}} (\nabla F_k(\hat{\boldsymbol{\theta}})_j)^2$ in the following Lemma.

Lemma 4. *For $\lambda \leq M$, $\sum_{k=1}^{M} (\nabla F_k(\hat{\boldsymbol{\theta}}))_j)^2$ has the following upper bound:*

$$\sum_{k=1}^{M} (\nabla F_k(\hat{\boldsymbol{\theta}})_j)^2 \leq 2\left(\|(\mathbf{P}(\hat{\boldsymbol{\theta}}) - \mathbf{I})\mathbf{y}\|_2^2 + \lambda\right).$$

Proof. See the supplementary material [1]. □

Now we provide the main theorem about the convergence properties of DF-DSCD.

Theorem 1. *In* DF-DSCD*, for any iteration, feature j, and $\hat{\boldsymbol{\theta}}$, there exists a step size η such that the expectation of the loss function of* DF-DSCD *decreases: i.e.,*

$$E[F(\hat{\boldsymbol{\theta}} + \Delta\hat{\boldsymbol{\theta}}) - F(\hat{\boldsymbol{\theta}})] < 0$$

Proof. See the supplementary material [1]. □

Note that the step size η can be any value satisfying the following condition (see [1] for details).

$$\eta < \frac{(\nabla F(\hat{\boldsymbol{\theta}})_j)^2}{c(1+\epsilon)\left(\|(\mathbf{P}(\hat{\boldsymbol{\theta}}) - \mathbf{I})\mathbf{y}\|_2^2 + \lambda\right)}. \quad (6)$$

4.2 Complexity Analysis

Table 3 shows the time complexity of DF-DSCD. Per iteration, each mapper requires $O(n/M)$ number of operations to split the original n data instances into M machines. The number of operations each reducer requires to update the assigned coordinates is $O(nsd/M)$ where s is the density and d is the dimension of the data. Thus, DF-DSCD for T iterations has the time complexity $O(T(n/M + nsd/M))$. Note that DF-DSCD runs linearly on the number of data instances and the dimension of the data. D-SGD has the same time complexity as DF-DSCD. However, the actual number of operations is smaller in the case of DF-DSCD because each machine only updates P parameters instead of d parameters in DF-DSCD, where $P << d$. Also, as we will show empirically in Section 5.3, DF-DSCD requires a smaller number of iterations to converge compared to D-SGD. Compared to D-Shotgun whose time complexity is $O(Tnsd)$, DF-DSCD runs M times faster.

Table 3: Time complexity of DF-DSCD and other methods. DF-DSCD's time complexity is lower than that of D-Shotgun by M times. Although DF-DSCD's time complexity is the same as that of D-SGD, the actual number of operations is smaller in the case of DF-DSCD because each machine only updates P parameters instead of d parameters in DF-DSCD, where $P << d$. Also, DF-DSCD converges more quickly than D-SGD (see Section 5.3).

DF-DSCD	**D-SGD**	**D-Shotgun**
$O(T(n/M + nsd/M))$	$O(T(n/M + nsd/M))$	$O(Tnsd)$

5. EXPERIMENTS

We perform experiments to answer the following questions:

- **Q1 (Scalability):** What is the scalability of DF-DSCD compared to other algorithms w.r.t the number of data instances, the number of features, and both of them?
- **Q2 (Convergence):** Does DF-DSCD converge faster than other algorithms?
- **Q3 (Parameter):** What is (empirically) the optimal number P of coordinates to update for DF-DSCD?

We present extensive experimental results for the questions. After describing the experimental setting in Section 5.1, we answer the first question **Q1** in Section 5.2: with synthetic data of various sizes, we compare scalability of DF-DSCD with those of D-SGD and D-Shotgun. The second question **Q2** is answered in Section 5.3: with real-world datasets, we compare likelihood convergence and accuracy improvement of DF-DSCD with those of D-SGD and D-Shotgun. Finally, the third question **Q3** is answered in Section 5.4: we evaluate the performance when the number P of coordinates to update in DF-DSCD changes.

5.1 Experimental Settings

We describe the settings: cluster, data, and algorithms.

Cluster.

We run experiments in a 60-node Hadoop cluster. Each node in the cluster has Intel Xeon E5620 2.4GHz CPU and 24GB memory.

Data.

We use both real-world and synthetic datasets listed in Table 4.

Scalability Experiments. To validate scalability in Section 5.2, we test our algorithms on synthetically generated binary datasets. We generate the synthetic data as similar as real world data such as Netflix [3] with different sizes of scales. The Netflix data provides a training dataset of 100,480,507 ratings (1.18 percent density) that 480,189 users gave to 17,770 movies. Similarly, we generate a synthetic data, which consists of 480,000 instances and 18,000 features with 1 percent of non-zero items where a value of each item is restricted between 0 and 1. Then, we change the number of instances and features both equally (**S1**), only instances (**S2**), and only features (**S3**). Each data with different orders of magnitude has

Table 4: The properties of real-world and synthetic dataset: number of instances (n), number of features (d), density (s), number of nonzero items (nz), and size in disk ($size$).

Real world dataset.

	n	d	s (%)	$size$ (MB)
KDDa	8,407,752	20,216,830	0.0002	2,607
KDDb	19,264,097	29,890,095	0.0001	5,011
News20	17,996	1,355,191	0.03	137
RCV1	20,242	47,236	0.5	36

Synthetic dataset: S1 (scale both instances and features), S2 (scale only instances), and S3 (scale only features). k=thousand, m=million.

	n	d	nz	$size$ (MB)
S1-8	$480k * 2^{-4}$	$18k * 2^{-4}$	$100m * 2^{-8}$	44
S1-4	$480k * 2^{-2}$	$18k * 2^{-2}$	$100m * 2^{-4}$	752
S1-0	$480k$	$18k$	$100m$	12,559
S1+4	$480k * 2^{+2}$	$18k * 2^{+2}$	$100m * 2^{+4}$	207,174
S2-8	$480k * 2^{-8}$	$18k$	$100m * 2^{-8}$	48
S2-4	$480k * 2^{-4}$	$18k$	$100m * 2^{-4}$	784
S2-0	$480k$	$18k$	$100m$	12,559
S2+4	$480k * 2^{+4}$	$18k$	$100m * 2^{+4}$	200,951
S3-8	$480k$	$18k * 2^{-8}$	$100m * 2^{-8}$	52
S3-4	$480k$	$18k * 2^{-4}$	$100m * 2^{-4}$	764
S3-0	$480k$	$18k$	$100m$	12,559
S3+4	$480k$	$18k * 2^{+4}$	$100m * 2^{+4}$	222,766

its own name for convenience. For example, when changing both instances and features in **S1**, we scale down/up the size of instances and features by 2^{-4} (*S1-8*), 2^{-2} (*S1-4*), 2^{0} (*S1-0*), and 2^{+2} (*S1+4*), respectively.

Convergence and Accuracy Experiments. To validate the likelihood convergence and accuracy in Sections 5.3 and 5.4, we evaluate our algorithms on real-world datasets: KDDa, KDDb, News20, and RCV1.

1. *KDDa/KDDb*: KDD Cup 2010 dataset obtained from Carnegie Learning and DataShop [27]. Very large number of instances (19,264,097) and features (29,890,095) with up to 0.0002 density.
2. *News20*: size-balanced two-class variant of the UCI 20 Newsgroups dataset [10]. Very large number of features (1,355,191) with 0.03% density.
3. *Reuters Corpus Volume I (RCV1)*: an archive of over 800,000 manually categorized newswire stories made by Reuters for research purposes [13]. Large number of instances (20,242) and large number of features (47,236) with 0.5% density.

For each real-world dataset, we used the training and testing sets given by the data provider. Each algorithm iterates T times until the difference between successive parameter values is less than a given threshold (e.g., 0.01). For step size, we use a fixed step size η for D-SGD and DF-DSCD. The C parameter for D-Shotgun, which denotes the number of coordinates updated per iteration, is set to the optimal value $d/(2\rho)$ as suggested in the paper [5].

Algorithms.

We compare DF-DSCD with the following competitors.

1. *Distributed Stochastic Gradient Descent (D-SGD)* that parallelizes over instances only, as described in Section 3.1. This is the MapReduce version of Parallel Stochastic Gradient Descent [28].
2. *Distributed Shotgun (D-Shotgun)* that parallelizes over features, as described in Section 3.2. This is the MapReduce version of Parallel Stochastic Coordinate Descent [5]. We use the optimal $C = d/(2\rho)$ as suggested in the paper.

When choosing the parameter P of DF-DSCD in Section 5.3, we use $P = d/M$ since it provides the best performance as we will describe in Section 5.4. We vary the parameter P in Section 5.4.

5.2 Scalability

We empirically show that DF-DSCD scales well with the size of data, dimension of the data, and the number of machines.

Data Scalability.

To evaluate data scalability, we increase both the size of the number of instances and the features (Figure 3 (a)), only the number of instances (Figure 3 (b)), and only the number of the features (Figure 3 (c)) of synthetic datasets, and measure the running time of all the methods. Note that D-Shotgun fails to finish the job for the large datasets due to the overhead of broadcasting the dataset to all the machines. DF-DSCD and D-SGD show similar scalability for relatively low-dimensional datasets (S1 and S2) as shown in Figure 3(a,b). However, DF-DSCD outperforms D-SGD by 1.45× for relatively high-dimensional dataset S3 as shown in Figure 3(c). Below, We will discuss this in detail.

Dimension Scalability.

DF-DSCD processes very high dimensional data effectively thanks to its feature parallelism which D-SGD lacks. In the previous paragraph we verified the claim on a synthetic dataset with relatively high dimension (Figure 3(c)). The performance difference becomes more evident for even higher dimensional data: see Figure 5(a,b,c) whose dimensions are 20 million, 29 million, and 1.3 million, resp. Note that only DF-DSCD successfully analyzes the data while others fail.

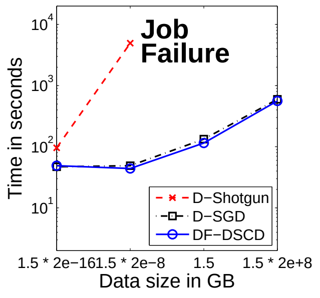
Machine Scalability.

Finally, we show that DF-DSCD scales almost linearly with the number of machines. We measure the speed-up by increasing the number of machines used on our Hadoop platform. To test on very large data, we run DF-DSCD a very large synthetic dataset **S1+4** (207 GB) that consists of 1.6 billions of non-zero items with 1.92 millions of instances and 72 thousands of features (see Table 4). Figure 4 shows the speed-up (throughput $1/T_M$, where T_M is the running time with M machines) of DF-DSCD by increasing the number of machines from 15, 30, 45 to 60. Note that DF-DSCD speeds up near linearly in the beginning, while the performance flattens as the number of machines increases, due to the overhead in distributed systems (e.g. JVM loading time, synchronization time, etc.).

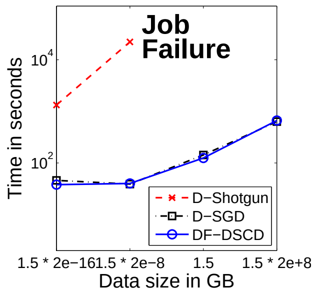
5.3 Convergence

We show that DF-DSCD converges faster than competitors. We also find the tendency that its performance becomes better as the number of coordinates updated by each machine becomes larger.

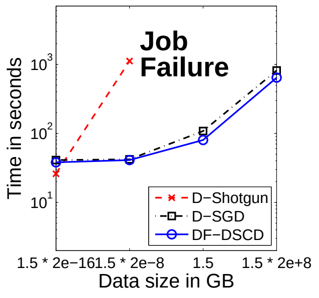
To evaluate how quickly DF-DSCD converges, we plot accuracy vs. time of all methods, where the accuracy is evaluated on testing set of each dataset. Figure 5 shows the experimental results of accuracy for the real world datasets: (a) KDDa, (b) KDDb, (c) News20, and (d) RCV1. For D-Shotgun, we choose $C = d/(2\rho)$ following the paper [5]; for DF-DSCD we choose the maximum



(a) Dataset **S1**: scales both instances and features

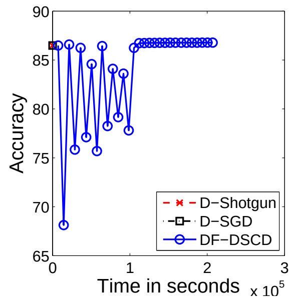


(b) Dataset **S2**: scales only instances

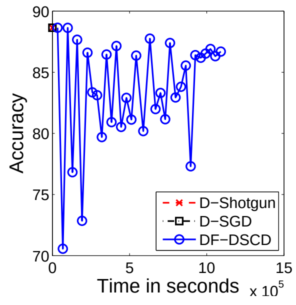


(c) Dataset **S3**: scales only features

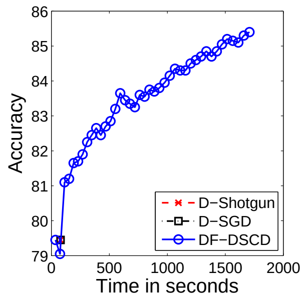
Figure 3: Scalability of DF-DSCD compared to D-SGD and D-Shotgun on four different scales of synthetic datasets. For all three datasets, DF-DSCD is the fastest. D-Shotgun fails on the two largest sizes of data in all the datasets due to heavy traffic overhead. Between DF-DSCD and D-SGD, DF-DSCD achieves better performance by partitioning data and features into different machines, while D-SGD partitions only data.



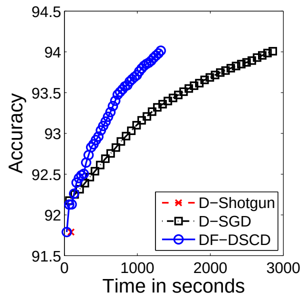
(a) **KDDa**



(b) **KDDb**



(c) **News20**



(d) **RCV1**

Figure 5: Accuracy vs. time in DF-DSCD, compared to D-SGD and D-Shotgun on real world datasets. Only DF-DSCD successfully handles all very high dimensional real-world datasets and converges the most rapidly in the lower dimensional dataset. D-Shotgun fails on the all datasets due to the overhead of data broadcasting. Comparing DF-DSCD with D-SGD, DF-DSCD performs well on KDDA, KDDb, and News20 data with very high dimensions, which could not be handled by D-SGD due to the requirements of updating all the coordinates. In RCV1 data, DF-DSCD is 2.2× faster than D-SGD to achieve the same accuracy.

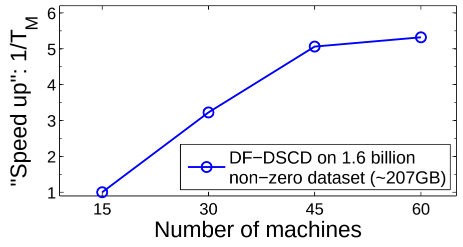


Figure 4: Runtime of DF-DSCD with different number of machines on a very large synthetic dataset **S1+4**. Note that DF-DSCD speeds-up near linearly in the beginning, while the performance flattens as the number of machines increases, due to the overhead in distributed systems (e.g. JVM loading time, synchronization time, etc.).

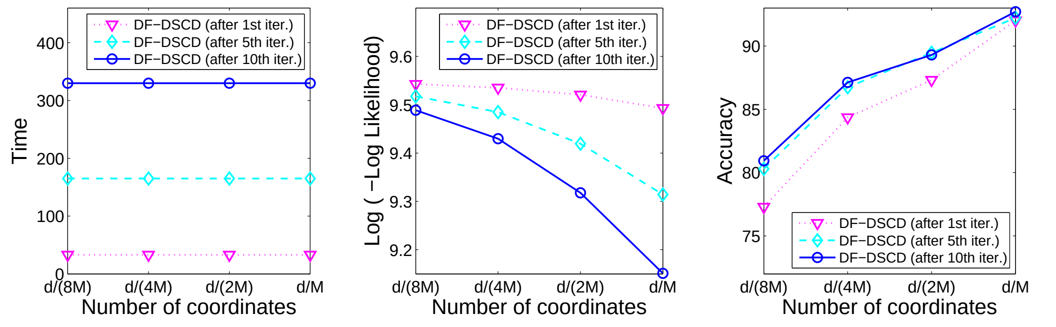
possible number $P = d/M$ based on the results from Section 5.4. Note that for all the results D-Shotgun fails to continue progress: the MapReduce job for D-Shotgun failed while working due to network overhead. While the original Shotgun [5] in multicore setting showed successful performance in accuracy, our result shows that it is intractable for Shotgun to work efficiently in MapReduce environment because all instances should be broadcast to all the machines.

Compared to D-SGD, DF-DSCD shows faster convergence than D-SGD to achieve the same accuracy. In KDDa, KDDb, and News20 (Figure 5 (a-c)) data, D-SGD fails: the reason is that KDDa, KDDb, and News20 have very large number of features (millions), and D-SGD needs to update all the coordinates. On the contrary, our DF-DSCD performs well in the high dimensional data since DF-DSCD can select a subset of the coordinates to update. Although D-SGD does not fail in RCV1 (Figure 5 (d)) data, DF-DSCD is 2.2× faster than D-SGD to get the same accuracy. In conclusion, the data parallelism allows DF-DSCD to work on large data instances which could not be handled by D-Shotgun, and the feature parallelism enables DF-DSCD to work on high dimensional data which could not be handled by D-SGD.

5.4 Effects of Number of Coordinates Updated

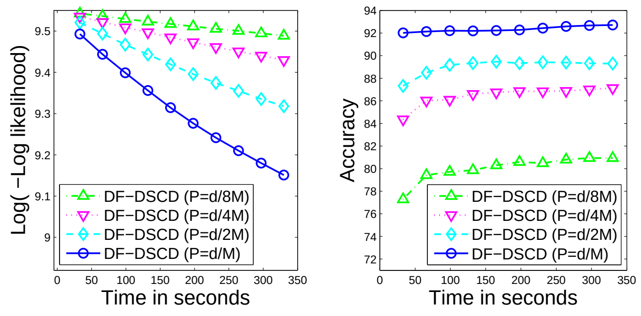
DF-DSCD requires a parameter P which is the number of coordinates to update per machine. The question is, which P provides the best performance? To answer the question, we vary P from $d/(8M)$, $d/(4M)$, $d/(2M)$ to the maximum possible value d/M, and compare the running time, likelihood, and accuracy of DF-DSCD on RCV1 data. Figures 6 and 7 show the result with different plotting scheme. Figure 6 shows the rate of (a) running time, (b) likelihood, and (c) accuracy over different number P of coordinates after 1^{st}, 5^{th}, and 10^{th} iterations. Figure 7 shows the rate of (a) likelihood convergence and (b) accuracy for different number of coordinates in DF-DSCD by plotting the likelihood or accuracy in y-axis against time (seconds) in x-axis.

In Figure 6 (a), notice that the running time is almost the same for different number P of coordinate to update per machine. This happens because the MapReduce algorithm for DF-DSCD heavily



(a) Running Time (b) Negative Likelihood (the lower the better) (c) Accuracy (the higher the better)

Figure 6: Comparison of (a) running time, (b) likelihood, (c) accuracy after 1^{st}, 5^{th}, and 10^{th} iterations in DF-DSCD with different number P of coordinates to update on RCV1 data. Larger number of updated coordinates per iteration leads to better performance in DF-DSCD. In (a), note that the running time is almost the same for different number P of coordinates since MapReduce algorithm for DF-DSCD heavily depends on disk accesses, and larger P increases only the CPU time which is small compared to the disk access time. In (b) and (c), DF-DSCD with $P = d/M$ shows the fastest convergence of likelihood and the highest accuracy after few initial iterations.



(a) Negative Likelihood (the lower the better) (b) Accuracy (the higher the better)

Figure 7: (a) Likelihood and (b) Accuracy vs. Running time of DF-DSCD with different number P of coordinate on RCV1 data. In concordance with our intuition, larger number of coordinates leads to better performance in DF-DSCD: $P = d/M$ provides the best negative log likelihood and the best accuracy for a given running time.

depends on disk accesses, and larger P increases mainly the CPU time which is small compared to the disk access time.

Figure 6 (b) shows that the negative likelihood decreases more quickly over iterations as the number P of coordinates to update increases. This means that if each machine uses the maximum number $P = d/M$ of coordinates to update, the data likelihood converges the most quickly. Figure 6 (c) shows the accuracy for different number P of coordinates to update per machine. As in the negative likelihood, the $P = d/M$ provides the best accuracy. Figure 7 (a) and (b) also show the similar result: data likelihood and accuracy is the best when choosing P as the maximum possible value d/M.

6. RELATED WORK

In this section, we review related works on parallel/distributed machine learning focusing on logistic regression.

Logistic Regression. Logistic regression is a widely-used method to solve classification problems in data mining. It has been applied to many applications such as life science [11], threat classification and temporal link analysis [7], anomaly detection [19], collaborative filtering [24] and text processing [17].

In the logistic regression problem, there have been several approaches of maximizing the likelihood of the entire dataset [9, 16]. However, the stochastic gradient descent and coordinate descent are the most famous approaches.

Parallelized/Distributed algorithms. Recently, parallel stochastic gradient descent algorithms for multicore [18] and distributed setting [15, 28] are studied. Especially, Zinkevich et al. [28] reduced I/O overhead of the algorithm by restricting training data to be accessed only locally and communicating at the very end. Even though data parallel stochastic gradient descent scales to large number of instances, it does not consider large number of features.

Bradley et al. [5] proposed Shotgun, a parallel coordinate descent algorithm for minimizing L_1-regularized losses. In the ideal case where all features are uncorrelated, Shotgun can do parallel updates up to the number of features. The algorithm is empirically proved to be one of the most scalable algorithms for L_1 minimization problem. Moreover, two preprocessing schemes [21,22] to improve Shotgun, which also can be easily applied to our DF-DSCD, are proposed. Although Shotgun provides feature parallelism, it does not provide data parallelism and its distributed algorithm.

Recently, Richtárik et al. [20] developed a distributed coordinate descent method that is similar with D-Shotgun algorithm in Section 3.2. Even though the authors partition big data into multiple machines, the paper does not include theoretical proof for convergence and empirical evidence. Alekh Agarwal et al. [2] proposed a tera-scale linear system which achieves fast convergence speed by combining an online learning algorithm with a batch one. Although their method is compatible with Hadoop, it does not follow MapReduce model and requires an additional communication infrastructure for effective communication between mappers.

Application beyond logistic regression. Besides logistic regression problem, other applications also can be solved using Stochastic Gradient Descent or Coordinate Descent. Gemulla et al. [6] proposed a matrix factorization algorithm using distributed stochastic gradient descent. Cho-Jui et al. [8] also used fast coordinate descent methods for variable selection in nonnegative matrix factorization. Recently, Hsiang-Fu et al. [25], proposed a coordinate descent based matrix factorization technique for recommendation system. Our DF-DSCD can be directly applied to solve other L_1 regularized loss minimization problem like Lasso. More broadly, our data and feature parallelism technique can be applied to other machine learning algorithms such as matrix/tensor factorization, linear SVM [26], large-margin learning [14], and conditional random field [4].

7. CONCLUSION

In this paper, we propose Data/Feature Distributed Stochastic Coordinate Descent (DF-DSCD), an efficient algorithm for solving logistic regression, or L_1 regularization in general, in a fully distributed way. The main contributions are the followings:

- **Design.** We carefully design DF-DSCD, a data/feature distributed stochastic coordinate descent algorithm for large scale logistic regression. DF-DSCD satisfies *both* data and feature parallelism which are two desired properties of distribution computation, while Distributed Stochastic Gradient Descent (D-SGD) and Distributed Shotgun (D-Shotgun) satisfies *either* data or feature parallelism, respectively.
- **Scalability.** DF-DSCD outperforms its competitors in terms of scalability. In terms of data size, DF-DSCD successfully analyzes 207 GB data while D-Shotgun fails to analyze it. In terms of the dimensionality of the data, DF-DSCD runs on the data with ~29 million features, while D-SGD fails to run on it.
- **Convergence.** From the theoretical analysis, we prove that DF-DSCD decreases the loss function of the logistic regression at every iteration. We also present experimental results showing that DF-DSCD converges up to 2.2× faster than its competitors.

Future works include extending DF-DSCD for other related tasks such as matrix factorization and tensor decomposition.

Acknowledgments

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT and Future Planning (grants No. 2013005259 and No. 2013R1A1A1064409).

8. REFERENCES

[1] `http://kdm.kaist.ac.kr/papers/dfdscd_supp_theory.pdf`.

[2] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *arXiv preprint arXiv:1110.4198*, 2011.

[3] J. Bennett, S. Lanning, and N. Netflix. The netflix prize. In *KDD Cup*, 2007.

[4] J. K. Bradley. *Learning Large-Scale Conditional Random Fields*. PhD thesis, CMU, 2013.

[5] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *ICML*, 2011.

[6] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, 2011.

[7] A. Goldenberg, J. Kubica, and P. Komarek. A comparison of statistical and machine learning algorithms on the task of link completion. In *KDD Workshop on Link Analysis for Detecting Complex Behavior*, 2003.

[8] C.-J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *KDD*, 2011.

[9] C. jen Lin, R. C. Weng, and S. S. Keerthi. Trust region newton method for large-scale logistic regression. In *ICML*, 2007.

[10] S. S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *JMLR*, 6, 2005.

[11] P. R. Komarek and A. W. Moore. Fast robust logistic regression for large sparse datasets with binary outputs. In *AISTATS*, 2003.

[12] S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient L_1 regularized logistic regression. In *AAAI*, 2006.

[13] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 2004.

[14] P. Long and R. Servedio. Algorithms and hardness results for parallel large margin learning. In *NIPS*, 2011.

[15] G. Mann, R. T. McDonald, M. Mohri, N. Silberman, and D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*, 2009.

[16] T. P. Minka. A comparison of numerical optimizers for logistic regression. Technical report, 2003.

[17] D. Nguyen, N. A. Smith, and C. P. Rosé. Author age prediction from text using linear regression. In *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, 2011.

[18] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011.

[19] H. Qiu, Y. Liu, N. A. Subrahmanya, and W. Li. Granger causality for time-series anomaly detection. In *ICDM*, 2012.

[20] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *arXiv:1310.2059*, 2013.

[21] C. Scherrer, M. Halappanavar, A. Tewari, and D. Haglin. Scaling up coordinate descent algorithms for large l_1 regularization problems. In *ICML*, 2012.

[22] C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin. Feature clustering for accelerating parallel coordinate descent. In *NIPS*, 2012.

[23] S. Shalev-Shwartz and A. Tewari. Stochastic methods for l_1 regularized loss minimization. In *ICML*, page 117, 2009.

[24] S. Vucetic and Z. Obradovic. Collaborative filtering using a regression-based approach. *Knowl. Inf. Syst.*, 7(1), Jan. 2005.

[25] C.-J. H. S. S. Yu, Hsiang-Fu and I. S. Dhillon. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, pages 1–27, 2013.

[26] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. *TKDD*, 2012.

[27] H.-F. Yu, H.-Y. Lo, H.-P. Hsieh, J.-K. Lou, T. G. McKenzie, J.-W. Chou, P.-H. Chung, C.-H. Ho, C.-F. Chang, Y.-H. Wei, et al. Feature engineering and classifier ensemble for kdd cup 2010. In *Proceedings of the KDD Cup 2010 Workshop*, pages 1–16, 2010.

[28] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, 2010.