SCOPE: Scalable Composite Optimization for Learning on Spark

Shen-Yi Zhao, Ru Xiang, Ying-Hao Shi, Peng Gao and **Wu-Jun Li**
National Key Laboratory for Novel Software Technology
Department of Computer Science and Technology, Nanjing University, China
{zhaosy,xiangr,shiyh,gaop}@lamda.nju.edu.cn, liwujun@nju.edu.cn

Abstract

Many machine learning models, such as logistic regression (LR) and support vector machine (SVM), can be formulated as composite optimization problems. Recently, many distributed stochastic optimization (DSO) methods have been proposed to solve the large-scale composite optimization problems, which have shown better performance than traditional batch methods. However, most of these DSO methods might not be scalable enough. In this paper, we propose a novel DSO method, called scalable composite optimization for learning (SCOPE), and implement it on the fault-tolerant distributed platform Spark. SCOPE is both computation-efficient and communication-efficient. Theoretical analysis shows that SCOPE is convergent with linear convergence rate when the objective function is strongly convex. Furthermore, empirical results on real datasets show that SCOPE can outperform other state-of-the-art distributed learning methods on Spark, including both batch learning methods and DSO methods.

Introduction

Many machine learning models can be formulated as composite optimization problems which have the following form with finite sum of some functions: $\min_{\mathbf{w}\in\mathbb{R}^d} P(\mathbf{w}) = \frac{1}{n}\sum_i^n f_i(\mathbf{w})$, where $\mathbf{w}$ is the parameter to learn (optimize), n is the number of training instances, and $f_i(\mathbf{w})$ is the loss function on the training instance i. For example, $f_i(\mathbf{w}) = \log(1+e^{-y_i\mathbf{x}_i^T\mathbf{w}}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$ in logistic regression (LR), and $f_i(\mathbf{w}) = \max\{0, 1-y_i\mathbf{x}_i^T\mathbf{w}\} + \frac{\lambda}{2}\|\mathbf{w}\|^2$ in support vector machine (SVM), where λ is the regularization hyper-parameter and $(\mathbf{x}_i, y_i)$ is the training instance i with $\mathbf{x}_i \in \mathbb{R}^d$ being the feature vector and $y_i \in \{+1, -1\}$ being the class label. Other cases like matrix factorization and deep neural networks can also be written as similar forms of composite optimization.

Due to its efficiency and effectiveness, stochastic optimization (SO) has recently attracted much attention to solve the composite optimization problems in machine learning (Xiao 2009; Bottou 2010; Duchi, Hazan, and Singer 2011; Schmidt, Roux, and Bach 2013; Johnson and Zhang 2013; Zhang, Mahdavi, and Jin 2013; Shalev-Shwartz and Zhang 2013; 2014; Lin, Lu, and Xiao 2014; Nitanda 2014). Existing SO methods can be divided into two categories. The first category is stochastic gradient descent (SGD) and its variants, such as stochastic average gradient (SAG) (Schmidt, Roux, and Bach 2013) and stochastic variance reduced gradient (SVRG) (Johnson and Zhang 2013), which try to perform optimization on the primal problem. The second category, such as stochastic dual coordinate ascent (SDCA) (Shalev-Shwartz and Zhang 2013), tries to perform optimization with the dual formulation. Many advanced SO methods, such as SVRG and SDCA, are more efficient than traditional batch learning methods in both theory and practice for large-scale learning problems.

Most traditional SO methods are sequential which means that the optimization procedure is not parallelly performed. However, with the increase of data scale, traditional sequential SO methods may not be efficient enough to handle large-scale datasets. Furthermore, in this big data era, many large-scale datasets are distributively stored on a cluster of multiple machines. Traditional sequential SO methods cannot be directly used for these kinds of distributed datasets. To handle large-scale composite optimization problems, researchers have recently proposed several parallel SO (PSO) methods for multi-core systems and distributed SO (DSO) methods for clusters of multiple machines.

PSO methods perform SO on a single machine with multi-cores (multi-threads) and a shared memory. Typically, synchronous strategies with locks will be much slower than asynchronous ones. Hence, recent progress of PSO mainly focuses on designing asynchronous or lock-free optimization strategies (Recht et al. 2011; Liu et al. 2014; Hsieh, Yu, and Dhillon 2015; J. Reddi et al. 2015; Zhao and Li 2016).

DSO methods perform SO on clusters of multiple machines. DSO can be used to handle extremely large problems which are beyond the processing capability of one single machine. In many real applications especially industrial applications, the datasets are typically distributively stored on clusters. Hence, DSO has recently become a hot research topic. Many DSO methods have been proposed, including distributed SGD methods from primal formulation and distributed dual formulation. Representative distributed SGD methods include PSGD (Zinkevich et al. 2010), BAVG-

M (Zhang, Wainwright, and Duchi 2012) and Splash (Zhang and Jordan 2015). Representative distributed dual formulations include DisDCA (Yang 2013), CoCoA (Jaggi et al. 2014) and CoCoA+ (Ma et al. 2015). Many of these methods provide nice theoretical proof about convergence and promising empirical evaluations. However, most of these DSO methods might not be scalable enough.

In this paper, we propose a novel DSO method, called scalable composite optimization for learning (SCOPE), and implement it on the fault-tolerant distributed platform Spark (Zaharia et al. 2010). SCOPE is both computation-efficient and communication-efficient. Empirical results on real datasets show that SCOPE can outperform other state-of-the-art distributed learning methods on Spark, including both batch learning methods and DSO methods, in terms of scalability.

Please note that some asynchronous methods or systems, such as Parameter Server (Li et al. 2014), Petuum (Xing et al. 2015) and the methods in (Zhang and Kwok 2014; Zhang, Zheng, and Kwok 2016), have also been proposed for distributed learning with promising performance. But these methods or systems cannot be easily implemented on Spark with the MapReduce programming model which is actually a bulk synchronous parallel (BSP) model. Hence, asynchronous methods are not the focus of this paper. We will leave the design of asynchronous version of SCOPE and the corresponding empirical comparison for future study.

SCOPE

Framework of SCOPE

SCOPE is based on a master-slave distributed framework, which is illustrated in Figure 1. More specifically, there is a master machine (called Master) and p ($p \geq 1$) slave machines (called Workers) in the cluster. These Workers are called Worker_1, Worker_2, $\cdots$, and Worker_p, respectively.

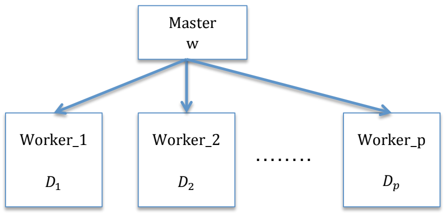


Figure 1: Distributed framework of SCOPE.

Data Partition and Parameter Storage

- For Workers: The whole dataset $\mathcal{D}$ is distributively stored on all the Workers. More specifically, $\mathcal{D}$ is partitioned into p subsets, which are denoted as $\{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_p\}$ with $\mathcal{D} = \bigcup_{k=1}^{p} \mathcal{D}_k$. $\mathcal{D}_k$ is stored on Worker_k. The data stored on different Workers are different from each other, which means that if $i \neq j$, $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$.
- For Master: The parameter $\mathbf{w}$ is stored on the Master and the Master always keeps the newest version of $\mathbf{w}$.

Algorithm 1 Task of Master in SCOPE

```
Initialization: $p$ Workers, $\mathbf{w}_0$;
for $t = 0, 1, 2, \ldots, T$ do
    Send $\mathbf{w}_t$ to the Workers;
    Wait until it receives $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_p$ from the $p$ Workers;
    Compute the full gradient $\mathbf{z} = \frac{1}{n}\sum_{k=1}^{p} \mathbf{z}_k$, and then send $\mathbf{z}$ to each Worker;
    Wait until it receives $\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2, \ldots, \tilde{\mathbf{u}}_p$ from the $p$ Workers;
    Compute $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p} \tilde{\mathbf{u}}_k$;
end for
```

Different Workers can not communicate with each other. This is similar to most existing distributed learning frameworks like MLlib (Meng et al. 2016), Splash, Parameter Server, and CoCoA and so on.

Optimization Algorithm The whole optimization (learning) algorithm is completed cooperatively by the Master and Workers:

- Task of Master: The operations completed by the Master are outlined in Algorithm 1. We can find that the Master has two main tasks. The first task is to compute the full gradient after all the *local gradient sum* $\{\mathbf{z}_k\}$ have been received from all Workers, and then send the full gradient to all Workers. The second task is to update the parameter $\mathbf{w}$ after all the *locally updated parameters* $\{\tilde{\mathbf{u}}_k\}$ have been received, and then send the updated parameter to all Workers. It is easy to see that the computation load of the Master is lightweight.
- Task of Workers: The operations completed by the Workers are outlined in Algorithm 2. We can find that each Worker has two main tasks. The first task is to compute the sum of the gradients on its local data (called *local gradient sum*), i.e., $\mathbf{z}_k = \sum_{i \in \mathcal{D}_k} \nabla f_i(\mathbf{w})$ for Worker_k, and then send the *local gradient sum* to the Master. The second task is to train $\mathbf{w}$ by only using the local data, after which the Worker will send the *locally updated parameters*, denoted as $\tilde{\mathbf{u}}_k$ for Worker_k, to the Master and wait for the newest $\mathbf{w}$ from Master.

Here, $\mathbf{w}_t$ denotes the global parameter at the tth iteration and is stored on the Master. $\mathbf{u}_{k,m}$ denotes the local parameter at the mth iteration on Worker_k.

SCOPE is inspired by SVRG (Johnson and Zhang 2013) which tries to utilize full gradient to speed up the convergence of stochastic optimization. However, the original SVRG in (Johnson and Zhang 2013) is sequential. To design a distributed SVRG method, one natural strategy is to adapt the mini-batch SVRG (Zhao et al. 2014) to distributed settings, which is a typical strategy in most distributed SGD frameworks like Parameter Server (Li et al. 2014) and Petuum (Xing et al. 2015). In appendix[1], we briefly outline the sequential SVRG and the mini-batch based distributed SVRG (called DisSVRG). We can find that there ex-

[1]All the appendices and proofs of this paper can be found in the arXiv version of this paper (Zhao et al. 2016).

Algorithm 2 Task of Workers in SCOPE

```
Initialization: initialize $\eta$ and $c > 0$;
For the Worker_$k$:
for $t = 0, 1, 2, \ldots, T$ do
    Wait until it gets the newest parameter $\mathbf{w}_t$ from the Master;
    Let $\mathbf{u}_{k,0} = \mathbf{w}_t$, compute the local gradient sum $\mathbf{z}_k = \sum_{i \in \mathcal{D}_k} \nabla f_i(\mathbf{w}_t)$, and then send $\mathbf{z}_k$ to the Master;
    Wait until it gets the full gradient $\mathbf{z}$ from the Master;
    for $m = 0$ to $M - 1$ do
        Randomly pick up an instance with index $i_{k,m}$ from $\mathcal{D}_k$;
        $\mathbf{u}_{k,m+1} = \mathbf{u}_{k,m} - \eta(\nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \mathbf{z} + c(\mathbf{u}_{k,m} - \mathbf{w}_t))$;
    end for
    Send $\mathbf{u}_{k,M}$ or $\frac{1}{M}\sum_{m=1}^{M} \mathbf{u}_{k,m}$, which is called the locally updated parameter and denoted as $\tilde{\mathbf{u}}_k$, to the Master;
end for
```

ist three major differences between SCOPE and SVRG (or DisSVRG).

The first difference is that in SCOPE each Worker *locally* performs stochastic optimization by only using its native data (refer to the update on $\mathbf{u}_{k,m+1}$ for each Worker_k in Algorithm 2). On the contrary, SVRG or DisSVRG perform stochastic optimization on the Master (refer to the update on $\mathbf{u}_{m+1}$) based on the whole dataset, which means that we need to randomly pick up an instance or a mini-batch from the whole dataset $\mathcal{D}$ in each iteration of stochastic optimization. The *locally stochastic optimization* in SCOPE can dramatically reduce the communication cost, compared with DisSVRG with mini-batch strategy.

The second difference is the update rule of $\mathbf{w}_{t+1}$ in the Master. There are no *locally updated parameters* in DisSVRG with mini-batch strategy, and hence the update rule of $\mathbf{w}_{t+1}$ in the Master for DisSVRG can not be written in the form of Algorithm 1, i.e., $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p} \tilde{\mathbf{u}}_k$.

The third difference is the update rule for $\mathbf{u}_{k,m+1}$ in SCOPE and $\mathbf{u}_{m+1}$ in SVRG or DisSVRG. Compared to SVRG, SCOPE has an extra term $c(\mathbf{u}_{k,m} - \mathbf{w}_t)$ in Algorithm 2 to guarantee convergence, where $c > 0$ is a parameter related to the objective function. The strictly theoretical proof will be provided in the following section about convergence. Here, we just give some intuition about the extra term $c(\mathbf{u}_{k,m} - \mathbf{w}_t)$. Since SCOPE puts no constraints about how to partition training data on different Workers, the data distributions on different Workers may be totally different from each other. That means the local gradient in each Worker can not necessarily approximate the full gradient. Hence, the term $\nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \mathbf{z}$ is a bias estimation of the full gradient. This is different from SVRG whose stochastic gradient is an unbias estimation of the full gradient. The bias estimation $\nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \mathbf{z}$ in SCOPE may lead $\mathbf{u}_{k,m+1}$ to be far away from the optimal value $\mathbf{w}^*$. To avoid this, we use the technique in the proximal stochastic gradient that adds an extra term $c(\mathbf{u}_{k,m} - \mathbf{w}_t)$ to make $\mathbf{u}_{k,m+1}$ not be far away from $\mathbf{w}_t$. If $\mathbf{w}_t$ is close to $\mathbf{w}^*$, $\mathbf{u}_{k,m+1}$ will also be close to $\mathbf{w}^*$. So the extra term in SCOPE is reasonable for convergence guarantee. At the same time, it does not bring extra computation since the update rule in SCOPE can be rewritten as

$$\begin{aligned}\mathbf{u}_{k,m+1} =& (1 - c\eta)\mathbf{u}_{k,m} \\ &- \eta(\nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \hat{\mathbf{z}}),\end{aligned}$$

where $\hat{\mathbf{z}} = \mathbf{z} - c\mathbf{w}_t$ can be pre-computed and fixed as a constant for different m.

Besides the above mini-batch based strategy (DisSVRG) for distributed SVRG, there also exist some other distributed SVRG methods, including DSVRG (Lee et al. 2016), KroMagnon (Mania et al. 2015), SVRGfoR (Konecný, McMahan, and Ramage 2015) and the distributed SVRG in (De and Goldstein 2016). DSVRG needs communication between Workers, and hence it cannot be directly implemented on Spark. KroMagnon focuses on asynchronous strategy, which cannot be implemented on Spark either. SVRGfoR can be implemented on Spark, but it provides no theoretical results about the convergence. Furthermore, SVRGfoR is proposed for cases with unbalanced data partitions and sparse features. On the contrary, our SCOPE can be used for any kind of features with theoretical guarantee of convergence. Moreover, in our experiment, we find that our SCOPE can outperform SVRGfoR. The distributed SVRG in (De and Goldstein 2016) cannot be guaranteed to converge because it is similar to the version of SCOPE with $c = 0$.

EASGD (Zhang, Choromanska, and LeCun 2015) also adopts a parameter like c to control the difference between the local update and global update. However, EASGD assumes that each worker has access to the entire dataset while SCOPE only requires that each worker has access to a subset. Local learning strategy is also adopted in other problems like probabilistic logic programs (Riguzzi et al. 2016).

Communication Cost

Traditional mini-batch based distributed SGD methods, such as DisSVRG in the appendix, need to transfer parameter $\mathbf{w}$ and stochastic gradients frequently between Workers and Master. For example, the number of communication times is $O(TM)$ for DisSVRG. Other traditional mini-batch based distributed SGD methods have the same number of communication times. Typically, $M = \Theta(n)$. Hence, traditional mini-batch based methods have $O(Tn)$ number of communication times, which may lead to high communication cost.

Most training (computation) load of SCOPE comes from the inner loop of Algorithm 2, which is done at local Worker without any communication. It is easy to find that the number of communication times in SCOPE is $O(T)$, which is dramatically less than $O(Tn)$ of traditional mini-batch based distributed SGD or distributed SVRG methods. In the following section, we will prove that SCOPE has a linear convergence rate in terms of the iteration number T. It means that to achieve an ϵ-optimal solution[2], $T = O(\log \frac{1}{\epsilon})$.

[2] $\hat{\mathbf{w}}$ is called an ϵ-optimal solution if $\mathbb{E}\|\hat{\mathbf{w}} - \mathbf{w}^*\|^2 \leq \epsilon$ where $\mathbf{w}^*$ is the optimal solution.

Hence, T is typically not large for many problems. For example, in most of our experiments, we can achieve convergent results with $T \leq 10$. Hence, SCOPE is communication-efficient. SCOPE is a synchronous framework, which means that some waiting time is also needed for synchronization. Because the number of synchronization is also $O(T)$, and T is typically a small number. Hence, the waiting time is also small.

SCOPE on Spark

One interesting thing is that the computing framework of SCOPE is quite suitable for the popular distributed platform Spark. The programming model underlying Spark is MapReduce, which is actually a BSP model. In SCOPE, the task of Workers that computes *local gradient sum* $\mathbf{z}_k$ and the training procedure in the inner loop of Algorithm 2 can be seen as the Map process since both of them only use local data. The task of Master that computes the average for both *full gradient* $\mathbf{z}$ and $\mathbf{w}_{t+1}$ can be seen as the Reduce process.

The MapReduce programming model is essentially a synchronous model, which need some synchronization cost. Fortunately, the number of synchronization times is very small as stated above. Hence, both communication cost and waiting time are very small for SCOPE. In this paper, we implement our SCOPE on Spark since Spark has been widely adopted in industry for big data applications, and our SCOPE can be easily integrated into the data processing pipeline of those organizations using Spark.

Convergence of SCOPE

In this section, we will prove the convergence of SCOPE when the objective functions are strongly convex. We only list some Lemmas and Theorems, the detailed proof of which can be found in the appendices (Zhao et al. 2016).

For convenience, we use $\mathbf{w}^*$ to denote the optimal solution. $\|\cdot\|$ denotes the L_2 norm $\|\cdot\|_2$. We assume that $n = pq$, which means that each Worker has the same number of training instances and $|\mathcal{D}_1| = |\mathcal{D}_2| = \cdots = |\mathcal{D}_p| = q$. In practice, we can not necessarily guarantee that these $|\mathcal{D}_k|$s are the same. However, it is easy to guarantee that $\forall i, j, |(|\mathcal{D}_i| - |\mathcal{D}_j|)| \leq 1$, which will not affect the performance.

We define p local functions as $F_k(\mathbf{w}) = \frac{1}{q}\sum_{i\in\mathcal{D}_k} f_i(\mathbf{w})$, where $k = 1, 2, \ldots, p$. Then we have $P(\mathbf{w}) = \frac{1}{p}\sum_{k=1}^{p} F_k(\mathbf{w})$.

To prove the convergence of SCOPE, we first give two assumptions which have also been widely adopted by most existing stochastic optimization algorithms for convergence proof.

Assumption 1 (Smooth Gradient)**.** *There exists a constant $L > 0$ such that $\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ and $i = 1, 2, \ldots, n$, we have $\|\nabla f_i(\mathbf{a}) - \nabla f_i(\mathbf{b})\| \leq L\|\mathbf{a} - \mathbf{b}\|$.*

Assumption 2 (Strongly Convex)**.** *For each local function $F_k(\cdot)$, there exists a constant $\mu > 0$ such that $\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, we have $F_k(\mathbf{a}) \geq F_k(\mathbf{b}) + \nabla F_k(\mathbf{b})^T(\mathbf{a} - \mathbf{b}) + \frac{\mu}{2}\|\mathbf{a} - \mathbf{b}\|^2$.*

Please note that these assumptions are weaker than those in (Zhang and Jordan 2015; Ma et al. 2015; Jaggi et al. 2014), since we do not need each $f_i(\mathbf{w})$ to be convex and we do not make any assumption about the Hessian matrices either.

Lemma 1. *Let $\gamma_m = \frac{1}{p}\sum_{k=1}^{p}\mathbb{E}\|\mathbf{u}_{k,m} - \mathbf{w}^*\|^2$. If $c > L - \mu$, then we have $\gamma_{m+1} \leq [1 - \eta(2\mu + c)]\gamma_m + (c\eta + 3L^2\eta^2)\gamma_0$.*

Let $\alpha = 1 - \eta(2\mu + c)$, $\beta = c\eta + 3L^2\eta^2$. Given L and μ which are determined by the objective function, we can always guarantee $0 < \alpha < 1$, $0 < \beta < 1$, and $\alpha + \beta < 1$ by setting $\eta < \min\{\frac{2\mu}{3L^2}, \frac{1}{2\mu+c}\}$. We have the following theorems:

Theorem 1. *If we take $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p}\mathbf{u}_{k,M}$, then we can get the following convergence result:*

$$\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq (\alpha^M + \frac{\beta}{1-\alpha})\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2.$$

When $M > \log_{\alpha}^{\frac{1-\alpha-\beta}{1-\alpha}}$, $\alpha^M + \frac{\beta}{1-\alpha} < 1$, which means we can get a linear convergence rate if we take $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p}\mathbf{u}_{k,M}$.

Theorem 2. *If we take $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p}\tilde{\mathbf{u}}_k$ with $\tilde{\mathbf{u}}_k = \frac{1}{M}\sum_{m=1}^{M}\mathbf{u}_{k,m}$, then we can get the following convergence result:*

$$\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq (\frac{1}{M(1-\alpha)} + \frac{\beta}{1-\alpha})\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2.$$

When $M > \frac{1}{1-\alpha-\beta}$, $\frac{1}{M(1-\alpha)} + \frac{\beta}{1-\alpha} < 1$, which means we can also get a linear convergence rate if we take $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p}\tilde{\mathbf{u}}_k$ with $\tilde{\mathbf{u}}_k = \frac{1}{M}\sum_{m=1}^{M}\mathbf{u}_{k,m}$.

According to Theorem 1 and Theorem 2, we can find that SCOPE gets a linear convergence rate when M is larger than some threshold. To achieve an ϵ-optimal solution, the computation complexity of each worker is $O((\frac{n}{p} + M)\log\frac{1}{\epsilon})$. In our experiment, we find that good performance can be achieved with $M = \frac{n}{p}$. Hence, SCOPE is computation-efficient.

Impact of Parameter c

In Algorithm 2, we need the parameter c to guarantee the convergence of SCOPE. Specifically, we need $c > L - \mu$ according to Lemma 1. Here, we discuss the necessity of c.

We first assume $c = 0$, and try to find whether Algorithm 2 will converge or not. It means that in the following derivation, we always assume $c = 0$.

Let us define another local function:

$$F_k^{(t)}(\mathbf{w}) = F_k(\mathbf{w}) + (\mathbf{z} - \nabla F_k(\mathbf{w}_t))^T(\mathbf{w} - \mathbf{w}^*)$$

and denote $\mathbf{w}_{k,t}^* = \arg\min_{\mathbf{w}} F_k^{(t)}(\mathbf{w})$.

Let $\mathbf{v}_{k,m} = \nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \mathbf{z} + c(\mathbf{u}_{k,m} - \mathbf{w}_t)$. When $c = 0$, $\mathbf{v}_{k,m} = \nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \mathbf{z}$. Then, we have $\mathbb{E}[\mathbf{v}_{k,m}|\mathbf{u}_{k,m}] = \nabla F_k^{(t)}(\mathbf{u}_{k,m})$ and $\nabla F_k^{(t)}(\mathbf{w}_t) = \mathbf{z}$. Hence, we can find that each local Worker actually tries to optimize the local function $F_k^{(t)}(\mathbf{w})$ with SVRG based on the local data $\mathcal{D}_k$. It means that if we set a relatively small η and a relatively large M, the $\mathbf{u}_{k,m}$ will converge to $\mathbf{w}^*_{k,t}$.

Since $F_k^{(t)}(\mathbf{w})$ is strongly convex, we have $\nabla F_k^{(t)}(\mathbf{w}^*_{k,t}) = 0$. Then, we can get

$$\nabla F_k(\mathbf{w}^*_{k,t}) - \nabla F_k(\mathbf{w}^*) = \nabla F_k(\mathbf{w}_t) - \nabla F_k(\mathbf{w}^*) - \mathbf{z}.$$

For the left-hand side, we have

$$\nabla F_k(\mathbf{w}^*_{k,t}) - \nabla F_k(\mathbf{w}^*) \approx \nabla^2 F_k(\mathbf{w}^*)(\mathbf{w}^*_{k,t} - \mathbf{w}^*).$$

For the right-hand side, we have

$$\begin{aligned}&\nabla F_k(\mathbf{w}_t) - \nabla F_k(\mathbf{w}^*) - \mathbf{z}\\ =&\nabla F_k(\mathbf{w}_t) - \nabla F_k(\mathbf{w}^*) - (\mathbf{z} - \nabla P(\mathbf{w}^*))\\ \approx&\nabla^2 F_k(\mathbf{w}^*)(\mathbf{w}_t - \mathbf{w}^*) - \nabla^2 P(\mathbf{w}^*)(\mathbf{w}_t - \mathbf{w}^*).\end{aligned}$$

Combining the two approximations, we can get

$$\mathbf{w}^*_{k,t} - \mathbf{w}^* \approx (\mathbf{I} - \mathbf{A}_k^{-1}\mathbf{A})(\mathbf{w}_t - \mathbf{w}^*),$$

where $\mathbf{A}_k = \nabla^2 F_k(\mathbf{w}^*)$ and $\mathbf{A} = \nabla^2 P(\mathbf{w}^*)$ are two Hessian matrices for the local function $F_k(\mathbf{w}^*)$ and the global function $P(\mathbf{w}^*)$, respectively. Assuming in each iteration we can always get the local optimal values for all local functions, we have

$$\mathbf{w}_{t+1} - \mathbf{w}^* \approx (\mathbf{I} - \frac{1}{p}\sum_{k=1}^{p} \mathbf{A}_k^{-1}\mathbf{A})(\mathbf{w}_t - \mathbf{w}^*). \quad (1)$$

Please note that all the above derivations assume that $c = 0$. From (1), we can find that Algorithm 2 will not necessarily converge if $c = 0$, and the convergence property is dependent on the Hessian matrices of the local functions.

Here, we give a simple example for illustration. We set $n = p = 2$ and $F_1(\mathbf{w}) = (\mathbf{w} - 1)^2, F_2(\mathbf{w}) = 100(\mathbf{w} - 10)^2$. We set a small step-size $\eta = 10^{-5}$ and a large $M = 4000$. The convergence results of SCOPE with different c are presented in Table 1.

Table 1: Impact of c.

c	0	1	5	10
Converge?	No	No	No	Yes

Separating Data Uniformly

If we separate data uniformly, which means that the local data distribution on each Worker is similar to the global data distribution, then we have $\mathbf{A}_k \approx \mathbf{A}$ and $\|\mathbf{I} - \frac{1}{p}\sum_{i=1}^{p} \mathbf{A}_k^{-1}\mathbf{A}\| \approx 0$. From (1), we can find that $c = 0$ can make SCOPE converge for this special case.

Experiment

We choose logistic regression (LR) with a L_2-norm regularization term to evaluate SCOPE and baselines. Hence, $P(\mathbf{w})$ is defined as $P(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left[\log(1 + e^{-y_i \mathbf{x}_i^T \mathbf{w}}) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right]$. The code can be downloaded from `https://github.com/LIBBLE/LIBBLE-Spark/`.

Dataset

We use four datasets for evaluation. They are MNIST-8M, epsilon, KDD12 and Data-A. The first two datasets can be downloaded from the LibSVM website[3]. MNIST-8M contains 8,100,000 handwritten digits. We set the instances of digits 5 to 9 as positive, and set the instances of digits 0 to 4 as negative. KDD12 is the dataset of Track 1 for KDD Cup 2012, which can be downloaded from the KDD Cup website[4]. Data-A is a dataset from a data mining competition[5]. The information about these datasets is summarized in Table 2. All the data is normalized before training. The regularization hyper-parameter λ is set to 10^{-4} for the first three datasets which are relatively small, and is set to 10^{-6} for the largest dataset Data-A. Similar phenomenon can be observed for other λ, which is omitted due to space limitation. For all datasets, we set $c = \lambda \times 10^{-2}$.

Table 2: Datasets for evaluation.

	♯instances	♯features	memory	λ
MNIST-8M	8,100,000	784	39G	1e-4
epsilon	400,000	2,000	11G	1e-4
KDD12	73,209,277	1,427,495	21G	1e-4
Data-A	106,691,093	320	260G	1e-6

Experimental Setting and Baseline

Distributed Platform We have a Spark cluster of 33 machines (nodes) connected by 10GB Ethernet. Each machine has 12 Intel Xeon E5-2620 cores with 64GB memory. We construct two clusters, a small one and a large one, from the original 33 machines for our experiments. The small cluster contains 9 machines, one master and eight slaves. We use 2 cores for each slave. The large cluster contains 33 machines, 1 master and 32 slaves. We use 4 cores for each slave. In both clusters, each machine has access to 64GB memory on the corresponding machine and one core corresponds to one Worker. Hence, the small cluster has one Master and 16 Workers, and the large cluster has one Master and 128 Workers. The small cluster is for experiments on the three relatively small datasets including MNIST-8M, epsilon and KDD12. The large cluster is for experiments on the largest dataset Data-A. We use Spark1.5.2 for our experiment, and implement our SCOPE in Scala.

Baseline Because the focus of this paper is to design distributed learning methods for Spark, we compare SCOPE with distributed learning baselines which can be implemented on Spark. More specifically, we adopt the following baselines for comparison:

- MLlib[6] (Meng et al. 2016): MLlib is an open source library for distributed machine learning on Spark. It is mainly based on two optimization methods: mini-batch based distributed SGD and distributed lbfgs. We find that

[3]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[4]http://www.kddcup2012.org/

[5]http://www.yiban.cn/project/2015ccf/comp_detail.php?cid=231

[6]http://spark.apache.org/mllib/

the distributed SGD method is much slower than distributed lbfgs on Spark in our experiments. Hence, we only compare our method with distributed lbfgs for MLlib, which is a batch learning method.

- LibLinear[7] (Lin et al. 2014): LibLinear is a distributed Newton method, which is also a batch learning method.
- Splash[8] (Zhang and Jordan 2015): Splash is a distributed SGD method by using the local learning strategy to reduce communication cost (Zhang, Wainwright, and Duchi 2012), which is different from the mini-batch based distributed SGD method.
- CoCoA[9] (Jaggi et al. 2014): CoCoA is a distributed dual coordinate ascent method by using local learning strategy to reduce communication cost, which is formulated from the dual problem.
- CoCoA+[10] (Ma et al. 2015): CoCoA+ is an improved version of CoCoA. Different from CoCoA which adopts average to combine local updates for global parameters, CoCoA+ adopts adding to combine local updates.

We can find that the above baselines include state-of-the-art distributed learning methods with different characteristics. All the authors of these methods have shared the source code of their methods to the public. We use the source code provided by the authors for our experiment. For all baselines, we try several parameter values to choose the best performance.

Efficiency Comparison with Baselines

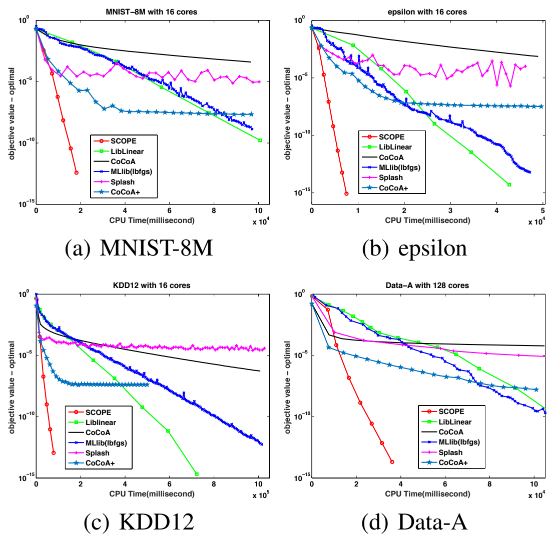
We compare SCOPE with other baselines on the four datasets. The result is shown in Figure 2. Each marked point on the curves denotes one update for $\mathbf{w}$ by the Master, which typically corresponds to an iteration in the outer-loop. For SCOPE, good convergence results can be got with number of updates (i.e., the T in Algorithm 1) less than five. We can find that Splash vibrates on some datasets since it introduces variance in the training process. On the contrary, SCOPE are stable, which means that SCOPE is a variance reduction method like SVRG. It is easy to see that SCOPE has a linear convergence rate, which also conforms to our theoretical analysis. Furthermore, SCOPE is much faster than all the other baselines.

SCOPE can also outperform SVRGfoR (Konecný, McMahan, and Ramage 2015) and DisSVRG. Experimental comparison can be found in appendix (Zhao et al. 2016).

Speedup

We use dataset MNIST-8M for speedup evaluation of SCOPE. Two cores are used for each machine. We evaluate speedup by increasing the number of machines. The training process will stop when the gap between the objective function value and the optimal value is less than 10^{-10}. The *speedup* is defined as follows: $speedup = \frac{time\ with\ 16\ cores\ by\ SCOPE}{time\ with\ 2\pi\ cores}$ where π is the number of machines and we choose $\pi = 8, 16, 24, 32$. The experiments are performed by 5 times and the average time is reported for the final speedup result.

[7]https://www.csie.ntu.edu.tw/∼ cjlin/liblinear/
[8]http://zhangyuc.github.io/splash
[9]https://github.com/gingsmith/cocoa
[10]https://github.com/gingsmith/cocoa



(a) MNIST-8M (b) epsilon
(c) KDD12 (d) Data-A

Figure 2: Efficiency comparison with baselines.

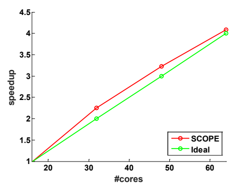


Figure 3: Speedup

The speedup result is shown in Figure 3, where we can find that SCOPE has a super-linear speedup. This might be reasonable due to the higher cache hit ratio with more machines (Yu et al. 2014). This speedup result is quite promising on our multi-machine settings since the communication cost is much larger than that of multi-thread setting. The good speedup of SCOPE can be explained by the fact that most training work can be locally completed by each Worker and SCOPE does not need much communication cost.

SCOPE is based on the synchronous MapReduce framework of Spark. One shortcoming of synchronous framework is the synchronization cost, which includes both communication time and waiting time. We also do experiments to show the low synchronization cost of SCOPE, which can be found in the appendix (Zhao et al. 2016).

Conclusion

In this paper, we propose a novel DSO method, called SCOPE, for distributed machine learning on Spark. Theoretical analysis shows that SCOPE is convergent with linear convergence rate for strongly convex cases. Empirical results show that SCOPE can outperform other state-of-the-art distributed methods on Spark.

Acknowledgements

This work is partially supported by the “DengFeng” project of Nanjing University.

References

Bottou, L. 2010. Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics*.

De, S., and Goldstein, T. 2016. Efficient distributed SGD with variance reduction. In *IEEE International Conference on Data Mining*.

Duchi, J. C.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12:2121–2159.

Hsieh, C.-J.; Yu, H.-F.; and Dhillon, I. S. 2015. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. In *International Conference on Machine Learning*.

J. Reddi, S.; Hefny, A.; Sra, S.; Poczos, B.; and Smola, A. J. 2015. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Neural Information Processing Systems*.

Jaggi, M.; Smith, V.; Takac, M.; Terhorst, J.; Krishnan, S.; Hofmann, T.; and Jordan, M. I. 2014. Communication-efficient distributed dual coordinate ascent. In *Neural Information Processing Systems*.

Johnson, R., and Zhang, T. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Neural Information Processing Systems*.

Konecný, J.; McMahan, B.; and Ramage, D. 2015. Federated optimization: Distributed optimization beyond the datacenter. arXiv:1511.03575.

Lee, J. D.; Lin, Q.; Ma, T.; and Yang, T. 2016. Distributed stochastic variance reduced gradient methods and a lower bound for communication complexity. arXiv:1507.07595v2.

Li, M.; Andersen, D. G.; Park, J. W.; Smola, A. J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E. J.; and Su, B. 2014. Scaling distributed machine learning with the parameter server. In *USENIX Symposium on Operating Systems Design and Implementation*.

Lin, C.-Y.; Tsai, C.-H.; Lee, C.-P.; and Lin, C.-J. 2014. Large-scale logistic regression and linear support vector machines using spark. In *IEEE International Conference on Big Data*.

Lin, Q.; Lu, Z.; and Xiao, L. 2014. An accelerated proximal coordinate gradient method. In *Neural Information Processing Systems*.

Liu, J.; Wright, S. J.; Ré, C.; Bittorf, V.; and Sridhar, S. 2014. An asynchronous parallel stochastic coordinate descent algorithm. In *International Conference on Machine Learning*.

Ma, C.; Smith, V.; Jaggi, M.; Jordan, M. I.; Richtárik, P.; and Takác, M. 2015. Adding vs. averaging in distributed primal-dual optimization. In *International Conference on Machine Learning*.

Mania, H.; Pan, X.; Papailiopoulos, D. S.; Recht, B.; Ramchandran, K.; and Jordan, M. I. 2015. Perturbed iterate analysis for asynchronous stochastic optimization. arXiv:1507.06970.

Meng, X.; Bradley, J.; Yavuz, B.; Sparks, E.; Venkataraman, S.; Liu, D.; Freeman, J.; Tsai, D.; Amde, M.; Owen, S.; Xin, D.; Xin, R.; Franklin, M. J.; Zadeh, R.; Zaharia, M.; and Talwalkar, A. 2016. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research* 17(34):1–7.

Nitanda, A. 2014. Stochastic proximal gradient descent with acceleration techniques. In *Neural Information Processing Systems*.

Recht, B.; Re, C.; Wright, S. J.; and Niu, F. 2011. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Neural Information Processing Systems*.

Riguzzi, F.; Bellodi, E.; Zese, R.; Cota, G.; and Lamma, E. 2016. Scaling structure learning of probabilistic logic programs by mapreduce. In *European Conference on Artificial Intelligence*.

Schmidt, M. W.; Roux, N. L.; and Bach, F. R. 2013. Minimizing finite sums with the stochastic average gradient. *CoRR* abs/1309.2388.

Shalev-Shwartz, S., and Zhang, T. 2013. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research* 14(1):567–599.

Shalev-Shwartz, S., and Zhang, T. 2014. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *International Conference on Machine Learning*.

Xiao, L. 2009. Dual averaging method for regularized stochastic learning and online optimization. In *Neural Information Processing Systems*.

Xing, E. P.; Ho, Q.; Dai, W.; Kim, J. K.; Wei, J.; Lee, S.; Zheng, X.; Xie, P.; Kumar, A.; and Yu, Y. 2015. Petuum: A new platform for distributed machine learning on big data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Yang, T. 2013. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Neural Information Processing Systems*.

Yu, Z.-Q.; Shi, X.-J.; Yan, L.; and Li, W.-J. 2014. Distributed stochastic ADMM for matrix factorization. In *International Conference on Conference on Information and Knowledge Management*.

Zaharia, M.; Chowdhury, M.; Franklin, M. J.; Shenker, S.; and Stoica, I. 2010. Spark: Cluster computing with working sets. In *USENIX Workshop on Hot Topics in Cloud Computing*.

Zhang, Y., and Jordan, M. I. 2015. Splash: User-friendly programming interface for parallelizing stochastic algorithms. *CoRR* abs/1506.07552.

Zhang, R., and Kwok, J. T. 2014. Asynchronous distributed ADMM for consensus optimization. In *International Conference on Machine Learning*.

Zhang, S.; Choromanska, A.; and LeCun, Y. 2015. Deep learning with elastic averaging SGD. In *Neural Information Processing Systems*.

Zhang, L.; Mahdavi, M.; and Jin, R. 2013. Linear convergence with condition number independent access of full gradients. In *Neural Information Processing Systems*.

Zhang, Y.; Wainwright, M. J.; and Duchi, J. C. 2012. Communication-efficient algorithms for statistical optimization. In *Neural Information Processing Systems*.

Zhang, R.; Zheng, S.; and Kwok, J. T. 2016. Asynchronous distributed semi-stochastic gradient optimization. In *AAAI Conference on Artificial Intelligence*.

Zhao, S.-Y., and Li, W.-J. 2016. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *AAAI Conference on Artificial Intelligence*.

Zhao, T.; Yu, M.; Wang, Y.; Arora, R.; and Liu, H. 2014. Accelerated mini-batch randomized block coordinate descent method. In *Neural Information Processing Systems*.

Zhao, S.-Y.; Xiang, R.; Shi, Y.-H.; Gao, P.; and Li, W.-J. 2016. SCOPE: scalable composite optimization for learning on Spark. *CoRR* abs/1602.00133.

Zinkevich, M.; Weimer, M.; Li, L.; and Smola, A. J. 2010. Parallelized stochastic gradient descent. In *Neural Information Processing Systems*.