Distributed and Asynchronous Stochastic Gradient Descent with Variance Reduction

Yuewei Ming, Yawei Zhao, Chengkun Wu, Kuan Li, Jianping Yin

Abstract—Stochastic Gradient Descent (SGD) with variance reduction techniques have been proved powerful to train parameters of various machine learning models. However, they cannot support the distributed systems trivially due to the intrinsic design. Although conventional studies such as PetuumSGD perform well for distributed machine learning tasks, they almost focus on the optimization of the communication protocol, which do not exploit the potential benefits of a specific machine learning algorithm. We analyze both the variance reduction technique and the asynchronous communication protocol in PetuumSGD, and propose a distributed version of variance reduced SGD named *DisSVRG*. Specifically, DisSVRG adopts the variance reduction technique to update the parameters of a model, and then shares those new learned parameters across machines in a cluster by using asynchronous communication protocol. Besides, we accelerate DisSVRG by using the learning rate with an accelerated factor. An adaptive sampling strategy is proposed in DisSVRG, which reduces much wait time during the iterations, and speed up the convergence of DisSVRG. Extensive empirical study verifies that DisSVRG converges faster than the state-of-the-art variant of SGD, and gains almost linear speedup in a cluster.

Index Terms—Stochastic gradient descent, Variance reduction, Asynchronous communication protocol, Distributed machine learning algorithms.

1 Introduction

Machine learning based applications such as image recognization [1], speech recognization [2] and text processing [3] have got a lot proliferation in the era of Big Data. Those underlying machine learning models are usually complex and big with a large number of parameters which can be trained or learned from a huge size of training data. For example, it is possible to train a large scale deep network which consists of millions and even billions of parameters by feeding it decades of terabytes training data. Furthermore, it is noting that most of the machine learning algorithms are iterative convergent, which means those algorithms need many rounds of iterative calculations to update the parameters in their models. Considering the complex model, the huge size of the training data and the massive computation, an efficient training method is vitally important for a large scale learning task.

Many machine learning algorithms can be described the optimization problem like (1).

$$\min F(\omega), \qquad F(\omega) = \frac{1}{n}\sum_{i=1}^{n} f_i(\omega) \tag{1}$$

, and the update rule is:

$$\omega_t = \begin{cases} \omega_{t-1} - \nabla F(\omega_{t-1}) \ \backslash\backslash the\ gradient\ descent \\ \omega_{t-1} - \nabla f_i(\omega_{t-1}) \ \backslash\backslash the\ stochastic\ gradient\ descent \end{cases} \tag{2}$$

. Here, ∇ stands for the derivation operation. $F(\omega)$ is generally called the loss function. ω represents the parameters in a model, that is, the parameters needed to be updated during the iterations. n means the size of training data. t represents it is the tth iteration. Such the loss function $F(\omega)$ can be minimized by updating the parameters iteratively, which is called the learning process. Conventionally, the gradient descent method is used to compute the global average gradient, i.e., $\nabla F(\omega_{t-1})$, and then uses it to update the parameters during an iteration. Since$\nabla F(\omega_{t-1})$ needs n derivations, which are time-consuming, the gradient descent method is not practical for a large scale learning task. An alternative approach is Stochastic Gradient Descent (SGD) and its variants. SGD randomly samples an instance from the training data, and then use it to compute the local gradient, i.e., $\nabla f_i(\omega_{t-1})$, instead of the global average gradient. The parameters are thus updated by using the local gradient. Since $\nabla f_i(\omega_{t-1})$ merely needs one derivation for the local gradient during an iteration, it is efficient for the large scale learning tasks. However, since f_i is randomly sampled, the variance exists between the local gradient $\nabla f_i(\omega_{t-1})$ and the global average gradient $\nabla F(\omega_{t-1})$, which slows the convergence of the loss function i.e., $F(\omega)$, of a machine learning algorithm. Especially, when the parameters are close to the optimal state, it is difficult to decrease the loss function due to the variance. Conventionally, the variance is reduced by using a decaying learning rate to update the parameters. That is, the value of the learning rate is decreased when the iteration proceeds. Although the variance can be reduced by the decaying learning rate, the small learning rate unavoidably leads to the slow convergence.

Recently, the SGD and its variants have been widely used to train the parameters of a model in distributed memory systems [4], [5], [6]. Those versions of SGD generally train and update parameters in a parameter-server system by using a cluster. The machines in the parameter-server system are categorised into servers and workers. The underlying calculations of the gradients are conducted by workers. Such those

• *Y. Ming, Y. Zhao, C. Wu, K. Li and J. Yin are with the College of Computer, National University of Defense Technology, Changsha 410073, P.R. China. E-mail: myw123@163.com, zhaoyawei09@nudt.edu.cn, wchknudt@163.com, li.kuan@163.com, and jpyin@nudt.edu.cn.*

updates will be then pushed to servers, and be aggregated on servers for updating the global parameters. Finally, those new learned global parameters will be shared with workers. Since there exists much communication between workers and servers, all the distributed versions of SGD like PetuumSGD focus on the optimization of communication [6]. In specific, PetuumSGD has proposed the asynchronous communication protocol denoted by Staleness Synchronous Protocol (SSP) to conduct communication across machines. SSP accelerates PetuumSGD than the Bulk Synchronous Protocol (BSP) on Hadoop or Spark significantly. Since SSP is designed for the general iterative convergent machine learning algorithms, it does not exploit the potential benefits of SGD to accelerate the iterative calculations. For example, PetuumSGD updates the parameters by using a decaying learning rate. The learning rate in PetuumSGD in the current iteration denoted by η will become 0.95η in the next iteration. When the parameters are close to the optimal state, the loss function is difficult to be decreased due to the extremely small learning rate. In a nutshell, even though PetuumSGD adopts SSP to optimize the communication between workers and servers, the decay learning rate slows the convergence.

Meanwhile, a new technique of the variance reduction is proposed to speed up the convergence of SGD [7], [8], [9]. Such the variance reduction technique reduces the variance of SGD, and keeps SGD converging at a constant rate. However, those versions of SGD are designed to be used in one machine instead of a cluster. When the amount of parameters or the size of training data is extremely huge so that they cannot be stored in a machine, the underlying variance reduction technique will not work. For instance, a deep network may have billions or even trillions of parameters, which cannot be stored in a machine. Therefore, such versions of SGD are difficult to train a large scale learning task, and handle a huge size of training data.

In this paper, We design a distributed and asynchronous version of variance reduced SGD denoted by DisSVRG for large scale machine learning tasks. It is noting that DisSVRG adopts the asynchronous communication protocol, i.e., Staleness Synchronous Protocol (SSP). In order to obtain a fast convergence, DisSVRG is accelerated by using a learning rate with an accelerated factor. It is unavoidable for the fast workers to spend much time on waiting the slow workers when performing iterative calculations in a cluster, which is called the "straggler problem". The straggler problem wastes much time for the fasters workers, thus leads to the slow convergence of a machine learning algorithm. In order to reduce the wait time, we propose an adaptive sampling strategy to alleviate the straggler problem. Specifically, we dynamically adjust the random sampling strategy during the iterations. When the worker is faster than other workers, it will sample more instances for the next iteration, which will take the faster worker more time to compute the local gradient. Thus, the slow workers have chance to catch up the faster works, and the wait time is reduced significantly. Finally, we conduct empirical studies on the HPC cluster named *Tianhe* on the National Supercomputing Center in Changsha. The performance evaluation verifies that DisSVRG outperforms the state-of-the-art version of SGD, and obtains approximately linear speedup in the cluster.

The rest of this paper is organized as follows. Section 2 outlines the related work of our work. Section 3 presents the assumptions and the overview of DisSVRG. Section 4 illustrates the details of DisSVRG. Section 5 highlights the optimization of DisSVRG. Section 6 discusses the major difference between our work and the previous studies. Section 7 shows the performance evaluation. Section 8 concludes the paper.

2 Related work

With the proliferation of data, a complex and big model can be learned by feeding it a huge size of training data. An approach for such a large scale learning is to use a cluster to train the parameters of the underlying model [4], [5], [6]. Dean et al. propose a version of asynchronous and distributed SGD denoted by *DownpourSGD* in a parameter-server system. DownpourSGD uses fully asynchronous communication protocol to conduct communication across machines, which cannot guarantee the convergence. To increase the robustness of DownpourSGD, the Adagrad adaptive learning rate procedure is adopted [10]. However, the adopted learning rate is decaying with the iterations, and thus leads to the slow convergence. Besides, Li et al. and Xing et al. propose an implementation of the parameter-server system respectively. The similar asynchronous communication protocol denoted by SSP is adopted in both of their systems to share the updates of the parameters across machines. SSP has been proved powerful in both theory and practice. However, SGD in [5] uses a constant learning rate without variance reduction technique, leading to slow convergence caused by the variance. SGD in [6], i.e.,PetuumSGD, adopts a decaying learning rate to reduce the variance, giving rise to slow convergence when the learning rate becomes small. Our version of the asynchronous and distributed SGD, i.e., DisSVRG, adopts SSP to implement the communication across machines, and uses the variance reduction technique to reduce variance as well.

The latest variance reduced SGD denoted by *SVRG* is adopted in [7], which is effective to reduce the variance of SGD. However, SVRG is a serial version, and designed to be run on a single machine. Thus, it is not suitable for a large scale learning task. Asynchronous and parallel versions of SVRG partially solve this problem [8], [9], [11], [12]. Such versions SVRG uses the lock-free method to update the parameters in parallel for multiple learning threads in a machine. However, the design of such work targets at a multicore system on a single machine. When the size of training data or the number of parameters in the model is huge so that they can not be stored in one machine, SVRG and those variants do not work immediately. Our proposed variance reduced SGD is designed for a cluster which is effective to perform the large scale learning tasks.

3 Assumptions and Overview

3.1 Assumptions

Conventionally, each $f_i(\omega)$ in the optimization problem 1 is a L-smooth function. That is, $\exists$ a non-negative L, and $\forall$ a and

Algorithm 1 DisSVRG

1: Initialize $\tilde{\omega}^0$%Pull the global parameters by all trainers from the servers.
2: **for** $s = 1, 2, ...$ **do** % Asynchronously update the parameters by all trainers.
3: $\tilde{\omega} = \tilde{\omega}^{s-1}$
4: $\mu = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\omega_i)$
5: $\omega_0 = \tilde{\omega}$
6: **for** $t = 1, 2, ...m$ **do**
7: randomly sample $i_t \in \{0, 1, 2, ..., m\}$
8: $v_t = \nabla f_{i_t}(\omega_{t-1}) - \nabla f_{i_t}(\tilde{\omega}) + \mu$ %Variance reduced gradient.
9: $\omega_t = \omega_{t-1} - \eta v_t$%Update the parameters with variance reduction gradient.
10: $\tilde{\omega}_s = \omega_m$%Push the new learned parameters to the servers, and aggregate them with the global parameters.

b, the following inequality holds.

$$f_i(a) \leq f_i(b) + \nabla f_i(b)^{\mathrm{T}}(a-b) + \frac{L}{2} \parallel a-b \parallel^2 \tag{3}$$

Besides, the loss function $F(\omega)$ in the optimization problem 1 is γ-strongly convex, which means that $\exists$ a non-negative γ, and $\forall$ a and b, the following inequality holds.

$$F(a) \geq F(b) + \nabla F(b)^{\mathrm{T}}(a-b) + \frac{\gamma}{2} \parallel a-b \parallel^2 \tag{4}$$

It is noting that these assumptions are not tight, and many machine learning algorithms are under these assumptions including regression and classification. These assumptions will be exploited to optimize our version of SGD in Section 5.

3.2 Overview

Our distributed and asynchronous SGD denoted by DisSVRG is presented in Algorithm 1. DisSVRG consists of three ingredients: the epochs of iterations (the outer *for* loop at Line 2), the random sampling strategy of the training data (the inner *for* loop at Line 6) and the update rule of the parameters (Lines 8-9). DisSVRG is started by the servers, and will be informed to all the workers by message passing. Since a worker has received the message from a server, it pulls a copy of the global parameters from a server, and begins conducting the calculations during the epoch. The random sampling strategy is conducted by the workers. When a worker randomly samples an instance from the training data, it computes the variance reduced gradient (Line 8), and updates the local parameters with the local gradient (Line 9). When the local parameters have been updated, the new learned parameters will be sent to a server. The inter-machine communication is conducted by the asynchronous communication protocol, i.e., SSP. The server receives these learned parameters and aggregates them. The aggregated parameters are the latest global parameters which will be sent to workers for the next iteration. The details of communication across machines and aggregation of parameters will be demonstrated in Section 4.

Algorithm 2 Server

1: Initialize $\tilde{\omega}^0$
2: **while** true **do**
3: **if** receive a pull request from the trainer p **then**
4: $e_p = p.epoch$
5: **if** all the trainers have finished $(e_p - \tau)$th epoch **then**
6: send a copy of the global parameters.
7: **if** receive a push request from the trainer p **then**
8: receive the new learned parameters from the trainer p, and aggregate them with the global parameters on the server.

4 System Implementation

DisSVRG is designed for the distributed memory system where the machines can be categorized into workers and servers. Every worker caches a copy of the parameters which is called the *local parameters*; while all the servers maintains one copy of the parameters which is called the *global parameters*. The global parameters will be pulled by a worker and be used as the initial parameters for an iteration. Meanwhile, such the initial parameters will replace the stale local parameters on the worker and become its new local parameters.

Server: The servers control the iterations by using the asynchronous communication protocol, i.e., SSP. Since the runtime environment of machines in a cluster varies a lot, the cost time of an epoch for different workers is not same. Therefore, there are fast and slow workers which conduct an epoch fast and slow respectively. It is noting that DisSVRG may not converge if all the workers update parameters in a fully asynchronous way. Therefore, we set a delay bound, i.e., τ. The delay τ is used to synchronize all the workers. For instance, when the fastest worker finishes the tth iteration, and the slowest worker does not finish the $(t-\tau)$th iteration, the fastest worker will be forced to stop and wait the slowest one. In specific, the fastest worker can not pull a copy of the global parameters from the servers, and thus has to wait for the slowest work. Until the slowest worker finishes the $(t-\tau)$th iteration, the fastest worker will re-start to conduct the iterations. When a server receives the new learned parameters from a worker, it will aggregated them with the global parameters on the server. After that, the latest parameters on the server will be pulled by the worker for the next iteration. The details are illustrated in Algorithm 2.

Worker: Workers in a cluster conduct machine learning tasks in asynchronous way. They pull the parameters from the servers by message passing. If a copy of the global parameters is pulled to the workers, those workers begin iterative calculations. During an epoch, workers first randomly pick an instance from the training data, then use the instance to compute the gradient. All the workers are independent with peers when conducting iterative calculations. When an epoch is finished, a worker has learned the new parameters, and will send those new learned parameters to a server. It is the servers that control when to synchronize among the workers.

Aggregation: When the workers push their new learned parameters to the servers, those parameters will be aggregated with the global parameters on the servers. The average be-

Algorithm 3 Trainer

```
while true do
    send a pull request to a server
    while true do
        if receive a copy of the global parameters from the server. then
            cache it as the new local parameters, i.e., $\tilde{\omega}$.
            $\mu = \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(\tilde{\omega})$
            for $t = 0, 1, 2, ..., m$ do
                randomly sample a non-negative number $i$ with $i \in \{0, 1, 2, ..., n\}$
                $v_t = \nabla f_{i_t}(\omega_{t-1}) - \nabla f_{i_t}(\tilde{\omega}) + \mu$
                $\omega_t = \omega_{t-1} - \eta v_t$
            send the new learned parameter $\omega_m$ to a server.
```

tween the new learned parameters and the global parameters will be identified as the latest global parameters, and will wait to be pushed to all the workers for the next iteration.

Update rule: As illustrated in Algorithm 3, DisSVRG is significantly different from the standard SGD because of the variance reduction technique. In the standard SGD, the update rule is shown as follows.

$$v_t = \nabla f_{i_t}(\omega_{t-1}) \tag{5}$$

v_t in the standard SGD is not the global gradient, which leads to variance, and slows the convergence of the loss function unavoidably. Instead, the variance reduction technique is used in DisSVRG effectively reduce the variance [7]. Considering the ith round of the iterations, since i_t is randomly picked, it holds that

$$\mathbb{E}\nabla f_{i_t}(\omega_{t-1}) = \frac{1}{n}\nabla F(\omega_{t-1}) \tag{6}$$

, and

$$\mathbb{E}(-\nabla f_{i_t}(\tilde{\omega}) + \mu) = -\frac{1}{n}\nabla F(\tilde{\omega}) + \mathbb{E}(\mu) = 0 \tag{7}$$

. Therefore,

$$\mathbb{E}(\nabla f_{i_t}(\omega_{t-1}) - \nabla f_{i_t}(\tilde{\omega}) + \mu) = \mathbb{E}(\nabla f_{i_t}(\omega_{t-1})) = \frac{1}{n}\nabla F(\omega_{t-1}) \tag{8}$$

. It is obvious that the variance of DisSVRG is reduced as expected. Unlike PetuumSGD and DownpourSGD, DisSVRG can converge fast without decaying the learning rate during the iterations. Benefiting from the variance reduction technique, DisSVRG can converge at a constant rate.

5 Optimization of Asynchronous and Distributed SGD

5.1 Learning rate with an accelerated factor

Generally, variance reduction technique accelerates machine learning algorithms by using a constant learning rate. Even though the constant learning rate performs well for L-smooth and γ-strongly convex objection function, some intrinsic properties can be exploited to accelerate the convergence of the machine learning algorithms.

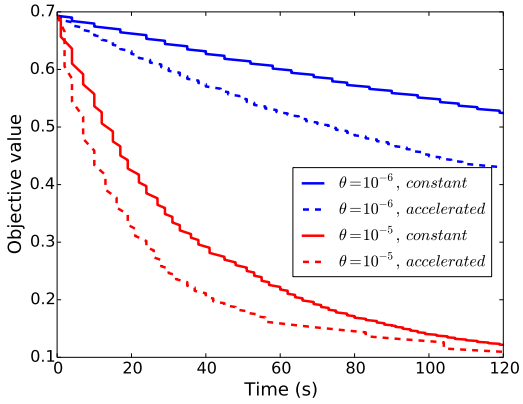


Fig. 1. The learning rate with the accelerated factor makes DisSVRG converge fast significantly.

We introduce an accelerated factor σ with $\sigma = \| \nabla f_i(\omega) \|$ to the learning rate of DisSVRG. That is,

$$\eta = \eta_0 + \sigma, \qquad \sigma = \| \nabla f_i(\omega) \| \tag{9}$$

. The learning rate of DisSVRG contains two ingredients: the constant and the accelerated factor. The constant part of the learning rate get the parameters out of the local optimal state; while the accelerated factor speedup the convergence of DisSGD. This setting of the accelerated factor is reasonable. First, the accelerated factor will become large when the parameters are far from the optimal state. If so, DisSVRG will converge fast benefiting from the large learning rate. Second, the accelerated factor will not become so large that DisSVRG do not converge. Considering that the convex function f_i in the machine learning model is L-smooth, ∇f_i will not be changed out of a range for an iteration. That is, $\| \nabla f_i \| < C$. Here, C is a non-negative constant. Third, when the parameters are close to the global optimal state, the accelerated factor δ will become close to zero. DisSVRG thus almost converges to the global optimal state at a constant just like the original design in [7].

As illustrated in Fig. 1, we use such the accelerated factor to conduct the linear regression tasks. Here, the evaluation test is conducted on a machine instead of a cluster. The dataset is *YearPredictMSD* which is the largest dataset we can find to conduct linear regression tasks. Other settings of the evaluation are presented in Section . We can get two important observations from Fig. 1. First, the learning rate with an accelerated factor makes the underlying machine learning algorithm converges faster than the constant learning rate without such the accelerated factor. Second, the benefits become significant with a small basic learning rate. Shortly, the accelerated factor gives rise to obvious benefits to accelerate the convergence of a machine learning algorithm.

5.2 Adaptive sampling strategy

Since DisSVRG needs the sampling strategy to update the parameters during an epoch, it is important to identify how many random updates in an epoch is appropriate. As illustrated in Algorithm 1, m represents the number of updates for an epoch. First, m cannot be given an extremely large number, which tasks much time to update the local parameters during

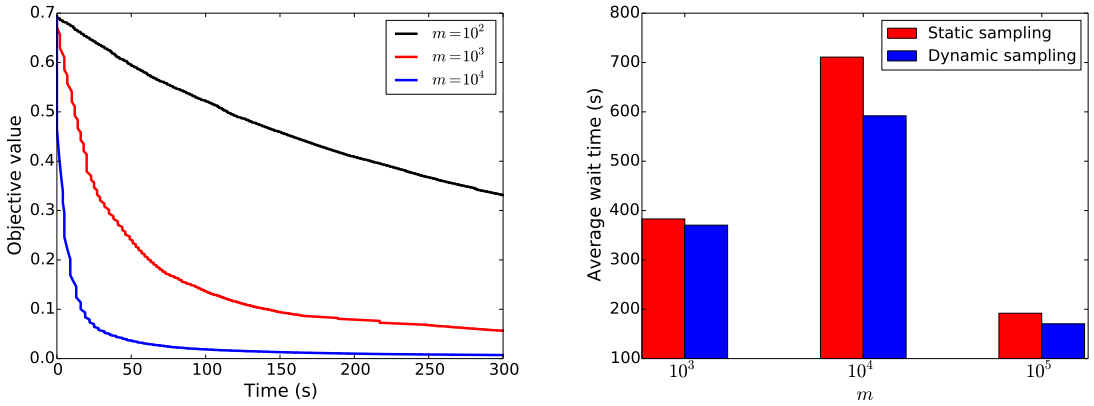


Fig. 2. A large m leads to a fast convergence for DisSVRG, and reduces much wait time during iterations.

an epoch on a worker. Second, m cannot be set a small value straightly. If so, DisSVRG has to conduct many epochs to reduce the value of the loss function. Considering that the average gradient of the loss function is needed to conduct an epoch, a small m means much computation of the average gradients, which is time-consuming.

Additionally, it is noting that the machines in a cluster perform differently due to the diversity of the runtime environment or the hardware in the heterogenous cluster. Although the asynchronous communication protocol, i.e., SSP, allows a delay bound to relax the synchronization among the machines, those fast machines have to wait the slow machines when the delay is achieved. Such wait time impairs the convergence of DisSVRG. An approach to reduce the underlying wait time is to adjust the sampling strategy dynamically. Intuitively, the fast machines in the cluster should sample more instances during an epoch than the slow machines. That is, m in the fast machines should be larger than that in the slow machines.

We adopt an adaptive dynamic sampling strategy which dynamically adjust m. When an epoch is completed, the new learned local parameters will be pushed to a server. The server will check the epochs of the worker. If the worker is too fast, it needs to wait other slow peers. The fast workers will adjust m to be a large value, i.e., $m+\delta$. Here, δ is a non-negative integer, and is set $\delta = 0.05m$ in DisSVRG. By using this adaptive strategy, the fast machines will sample more instances during an epoch, thus spend more time on computing the updates of parameters than the slow machines. The slow workers have chance to catch up the fast workers. Therefore, the wait time is reduced as a result. This adaptive sampling strategy has many benefits. The most important benefit is that the fast machines reduce the wait time, and use it to converge DisSVRG. We conduct an evaluation test on a machine by varying the value of m. Here, the dataset is still *YearPredictMSD*, and the learning rate, i.e., θ is set to a constant with $\theta = 10^{-5}$. As illustrated in Fig. 2(a), it is obvious that a large m brings a fast convergence of the objective function. Therefore, a large m benefits to decrease the value of the loss function. Additionally, the wait time is compared in a cluster which consists of 5 machines. As shown in Fig. 2(b), the average wait time has been evaluated by varying m. Here, the delay is set to 0. It is obvious that the adaptive sampling strategy decreases the average wait time during iterations, and thus spend much time to accelerate the convergence of DisSVRG in reverse.

6 Discussion

The machine learning tasks such as Deep learning are generally fed with extremely large volume of training data. However, conventional serial versions of SGD cannot handle a huge training data on a single machine within the available time. Although some distributed machine learning systems such as Petuum [6] and DMTK [13] have designed to solve this problem. Those variants of SGD have their inner weakness, namely the variance. Such those distributed machine learning systems thus decrease the learning rate to reduce the variance, which leads to slow convergence of SGD. We do not aim to proposing another a general platform for distributed machine learning algorithms, but focus on the optimization of SGD in a distributed system. Our implementation of the distributed SGD, i.e., DisSVRG, adopts the variance reduction technique to reduce the variance. Such the variance reduction technique is the most difference between DisSVRG and PetummSGD. Although PetuumSGD adopts a decayiny learning rate, it slows to converge the loss function. DisSVRG has been accelerated with two ingredients: the constant and the accelerated factor. The constant factor is effective to get DisSVRG out of the local optimal state, and keeps converge at a constant rate. The accelerated factor will speedup the convergence of the machine learning algorithm. Even though DisSVRG adopts the same asynchronous communication protocol with PetuumSGD, it converges faster than PetuumSGD by adopting the powerful variance technique.

DisSVRG is different from SVRG with at least three aspects. First, we extend the serial SVRG to an asynchronous and distributed version by using the asynchronous consistency protocol, i.e., SSP. Second, comparing with the constant learning rate in SVRG, the learning rate in our SGD contains an accelerated factor which exploits the potential benefits of the loss function, and makes the machine learning algorithms

converge fast. Third, SVRG needs to sample m instances randomly to update parameters. SVRG sets m multiple times of the size of training data, which is not practical for a large volume of training data. Instead, DisSVRG adopts an adaptive sampling strategy which adjusts the value of m by the runtime environment of the worker dynamically. Specifically, the fast workers will sample more instances during the next epoch than the slow peers. The slow workers thus have chance to catch up the fast workers. Thus, wait time is reduced significantly, which makes DisSVRG converge fast in the end. In a nutshell, considering the diversity of the runtime environment and the hardware in a cluster, DisSVRG is suitable to the practical scenarios.

Additionally, comparing with the version of SGD in [14] denoted by *SSGD*, our SGD adopts a more natural way to implement the distributed SGD with the variance reduction. SSGD implements the τ-delay bound inconsistent protocol within an epoch, but keeps fully consistent protocol among different workers. Therefore, SGD in [14] has at least two weaknesses. First, the fully consistency protocol for the workers is not suitable to the iterative convergence machine learning tasks [15], [16], [17]. Since the straggler problem usually exists among workers due to the variety of the system runtime environment or the hardware in the heterogenous cluster, the fast workers in [14] have to wait the slow workers, and start the next iteration in a synchronous way. Considering that machine learning algorithms are iterative convergent, such the fully consistency protocol wastes too much time. Second, SSGD is coupled with the hardware of clusters, which is not practical and natural. SSGD uses m learning threads to perform machine learning tasks where m is also the number of instances sampled in an epoch. That is, if the sampling strategy in SSGD adopts a large m, SSGD should be run in a cluster which can support m learning threads at a same time. Meanwhile, m in SSGD is set $O(n)$ where n represents the size of the training data. Considering the huge size of training data, m is really large in SSGD, which is not practical to a real cluster. In fact, a natural SGD should be designed to be flexible to work in different clusters by adjusting its settings. Instead, m in DisSVRG is identified by the adaptive sampling mechanism, which is flexible to be adjusted in the runtime environment. This design of the sampling strategy shields the difference of a specific cluster, which is more general and natural.

7 Performance Evaluation

In this section, we evaluate the performance of DisSVRG by using a regression problem and a classification problem on two datasets.

First we consider a regression problem:

$$\min \frac{1}{2n}\left(\frac{1}{1+e^{-\omega^{\mathrm{T}}x_i}}-y_i\right)^2 \tag{10}$$

Here, n is the size of training data. The dataset named *YearPredictMSD*[1] is the biggest dataset on the LibSVM for the regression problem. It contains 463715 samples, and each sample has 90 dimensions.

Additionally, we consider a classification problem:

$$\min -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log\frac{1}{1+e^{-\omega x_i}}+(1-y_i)\log(1-\frac{1}{1+e^{-\omega x_i}})\right) \tag{11}$$

Similarly, n is the size of training data. The datasets named *dna*[2] is used for the evaluation tests. Every sample in the dataset has 200 dimensions. The number of samples in the *dna* is $50,000,000$.

We conduct the evaluation test on the HPC cluster of the *Tianhe* supercomputer which is located in the National Supercomputing Center in Changsha. We have a maximum of 128 computing nodes, and each such node is equipped with two Intel Xeon X5670 CPUs and one Nvidia M2050 GPU. Each a CPU has 6 cores; while GPUs are not used in the evaluation test. m is set 1000. The learning rate, i.e., η is set 10^{-6}. For the fairness of the evaluation test, we use the third-party open source distributed machine learning system denoted by DMTK [13] to conduct the performance evaluation. All the algorithms which are used to be compared are implemented by the DMTK.

The following algorithms will be compared.

- **PetuumSGD:** The distributed version of SGD is implemented by using the asynchronous communication protocol, i.e., SSP [6]. The learning rate in PetuumSGD is decayed with a fixed factor 0.95 at the end of an epoch.
- **SSGD:** It is the state-of-the-art distributed version of SGD, which adopts the variance reduction technique [14]. The update rule in the SSGD has a variable θ which is used to update the parameters asynchronously. The details of SSGD can be referred in [14]. Here, we set $\theta = 0.5$.
- **DisSVRG-tricks:** DisSVRG is evaluated with all the optimization tricks.
- **DisSVRG-without-tricks:** DisSVRG is evaluated without any an optimization tricks.

7.1 Convergence

As illustrated in Fig. 3, the convergence performance of the algorithms has been evaluated. The delay is set 50 when those machine learning algorithms adopt the asynchronous communication protocol. All the datasets are handled on 32 workers. It is obvious that DisSVRG with the optimization tricks outperforms other algorithms for all the datasets. Even though DisSVRG does not use any an optimization trick, it always performs better than PetuumSGD, and gains a better performance than SSGD for the *dna* dataset. In specific, in order to decrease the loss function to 0.1, DisSVRG with all the optimization tricks spends one forth and one third time of the PetuumSGD for the datasets *YearPredictMSD* and *dna*, respectively. The main reason is that DisSVRG adopts asynchronous communication protocol as well as the variance reduction technique to update the parameters, thus better than any of the existing algorithms. Additionally, the learning rate with an accelerated factor speeds up the convergence. Meanwhile, the adaptive sampling strategy significantly reduces

[1] http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

[2] ftp://largescale.ml.tu-berlin.de/largescale

the underlying wait time which is used to accelerate the convergence of DisSVRG in reverse.

7.2 Speedup

As illustrated in Fig. 4, we compare the convergence performance of DisSVRG by varying the number of workers in a cluster. It is obvious that DisSVRG converges fast with a large number of workers. During the iterations, the more workers are used to update the local parameters, the more new learned parameters will be aggregated with the global parameters. After that, the global parameters which contains the new learned updates of parameters will be shared with other workers for the next iteration, thus accelerating the convergence of other workers. In specific, DisSVRG obtains appriximately linear speedup when varying the number of workers. For instance, DisSVRG gains $19\times$ speedup on the dataset *YearPredictMSD* when using 32 workers. DisSVRG even keeps the linear speedup when using 128 workers for the dataset *dna*. The approximately linear speedup mainly benefits from the asynchronous communication protocol and the variance reduction technique. The asynchronous communication protocol relaxes the bound of the synchronization among workers, which alleviates the straggler problem, and thus decreases the wait time. The variance reduction technique reduces the variance of SGD, and keeps DisSVRG converging at a constant rate.

7.3 Wait time

As shown in Fig. 5, we evaluate the average time consumption of DisSVRG by varying the value of the delay τ. It is obvious when the delay τ is small, the average wait time is large. A small delay means a tight bound among workers, which usually leads to the wait time for the fast workers due to the asynchronous communication protocol. For example, when the delay is set 0, all the workers should be synchronized for each iteration, thus waiting the most time. It is noting that the wait time decreases sharply when the delay becomes large. We concludes that a relax bound is effective to reduce the average wait time. Meanwhile, the average computing time will increase slightly with the relax bound of the delay. Although the fast workers have more freedom to conduct the iterations with the relax bound of the delay, the slow workers cannot obtain the new learned parameters from the fast workers within the large delay. The slow workers thus cannot benefit from the fast peers. In a nutshell, the delay should not be identified either too small or too large. It is a tradeoff between the wait time and the computing time for the workers. Generally, the delay should be set to minimize the total time consumption of DisSVRG. In our evaluation tests, the delay τ should be set 200 for the dataset *YearPredictMSD*, and 50 for the dataset *dna*.

8 Conclusion

Distributed SGD is an effective way to solve the large scale learning problems. In this paper, we propose a version of distributed SGD named DisSVRG with combing the asynchronous communication protocol and the variance reduction technique. Exploiting the properties of the loss function, DisSVRG is optimized by using a learning rate with the accelerated factor. Additionally, in order to reduce the wait time caused by the straggler problem, we propose an adaptive sampling strategy during the iterations. The adaptive sampling strategy gives the slow workers a chance to catch up the fast peers during iterations, thus alleviating the straggler problem. Extensive empirical studies show that DisSVRG converges faster than the state-of-the-art version of SGD, and can gain approximately linear speedup in a cluster.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Project NO. 60970034, 61170287, 61232016, 61303189 and 31501073).

References

[1] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," *Journal of Machine Learning Research*, vol. 15, pp. 215–223, 2011.

[2] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. ACM ICML*, 2008, pp. 160–167.

[4] J. Dean, G. Corrado, R. Monga, K. C. 0010, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks." in *Proc. NIPS*, 2012, pp. 1232–1240.

[5] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. UENSIX OSDI*, 2014, pp. 583–598.

[6] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: a new platform for distributed machine learning on big data," *Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[7] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," *Advances in Neural Information Processing Systems*, pp. 315–323, 2013.

[8] S.-Y. Zhao and W.-J. Li, "Fast asynchronous parallel stochastic gradient decent," *arXiv:1508.05711*, 2015.

[9] S. J. Reddi, A. Hefny, S. Sra, B. Póczos, and A. J. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants," pp. 2629–2637, 2015.

[10] R. L. Cavalcante, I. Yamada, and B. Mulgrew, "An adaptive projected subgradient approach to learning in diffusion networks," *Transactions on Signal Processing*, vol. 57, no. 7, pp. 2762–2774, 2009.

[11] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan, "Perturbed iterate analysis for asynchronous stochastic optimization," *arXiv:1507.06970*, 2015.

[12] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2719–2727.

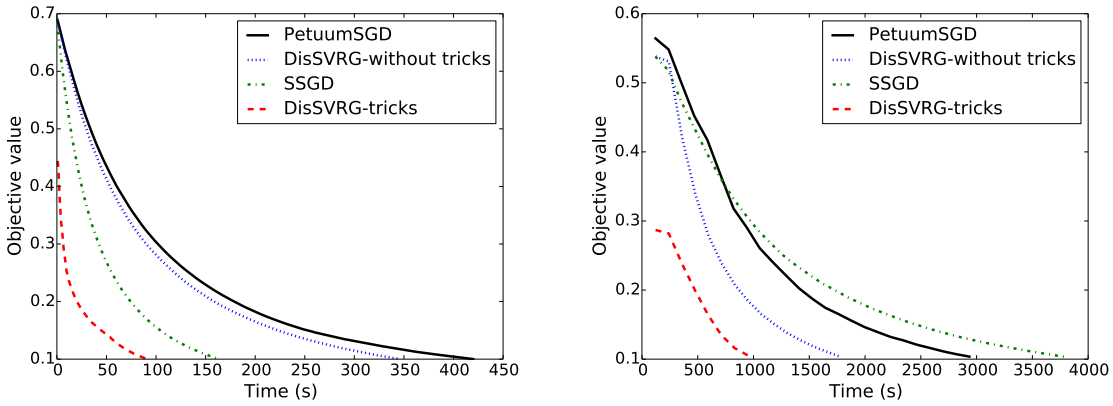
[13] J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng, E. P. Xing, T.-Y. Liu, and W.-Y. Ma, "Lightlda: Big topic models on modest computer clusters," in *Proc. WWW*, 2015, pp. 1351–1361.

[14] S. Z. Zhang, Ruiliang and J. T. Kwok, "Fast distributed asynchronous sgd with variance reduction," *arXiv:1508.01633*, 2015.

[15] W. Dai, A. Kumar, J. Wei, Q. Ho, G. A. Gibson, and E. P. Xing, "High-performance distributed ml at scale through parameter server consistency models," in *Proc. AAAI*, 2015, pp. 79–87.

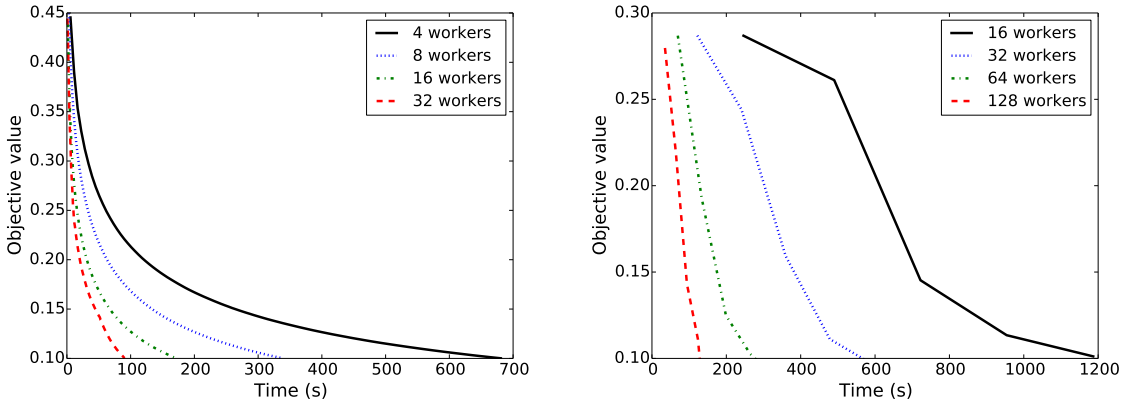
[16] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Proc. NIPS*, pp. 19–27, 2014.

[17] W. Dai, J. Wei, X. Zheng, J. K. Kim, S. Lee, J. Yin, Q. Ho, and E. P. Xing, "Petuum: A framework for iterative-convergent distributed ml," *arXiv:1312.7651*, 2013.



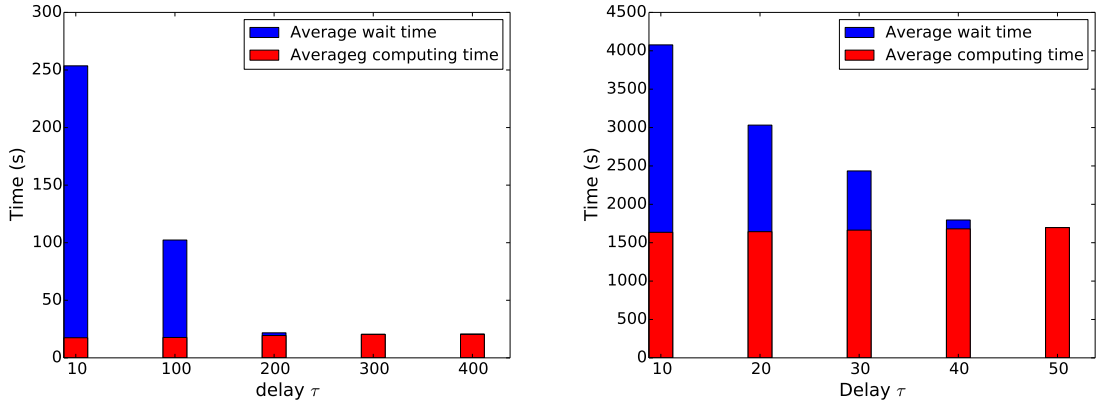
(a) *YearPredictMSD* (b) *dna*

Fig. 3. The performance of the convergence is compared by using 32 computing nodes.



(a) *YearPredictMSD* (b) *dna*

Fig. 4. DisSVRG obtains almost linear speedup when varying the number of workers.



(a) *YearPredictMSD* (b) *dna*

Fig. 5. The time consumption is compared by varying the delay τ for 128 workers.