Accelerate Distributed Stochastic Gradient Descent with Asynchronous Communication Protocol

Yuewei Ming, Yawei Zhao, Deke Guo *Member, IEEE,* Jianping Yin

Abstract—Asynchronous and distributed Stochastic Gradient Descent (SGD) with the variance reduction technique such as Stochastic Variance Reduction Gradient (SVRG) has been proved powerful to train various machine learning tasks. However, it cannot support the distributed machine learning algorithms directly because of excessive volume of training data in the era of Big Data. Although conventional work such as PetuumSGD performs well for distributed machine learning tasks, they focus on the optimization on communication protocols for iterative convergence algorithms which do not exploit the potential benefit of a specific machine learning algorithm. We analyze both SVRG and the asynchronous communication protocol in PetuumSGD, and propose a distributed implementation of SVRG by combining both of them together. Furthermore, we optimize our distributed SVRG with many tricks including an accelerated factor for the constant learning rate and adaptive sampling strategy to exploit the benefits SVRG. Extensive empirical study verifies that our distributed SVRG performs better on accelerating iterative convergence machine learning algorithms than pioneering work such as PetuumSGD.

Index Terms—Stochastic gradient descent, Staleness synchronous protocol, Distributed machine learning algorithms.

✦

1 INTRODUCTION

2 RELATED WORK

2.1 Asynchronous distributed stochastic gradient descent

3 PRELIMINARIES AND OVERVIEW

Many machine learning algorithms can be described by using

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

.

Here, n means the size of training data, and w represents the parameters of a machine learning model, that is, the parameters needed to be updated during the iterations. Conventionally, each $f_i(w)$ is a L-smooth function so that $\exists$ a non-negative L, and $\forall$ a and b, the following inequality holds.

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

.

The inequality can be described equivalently as follows.

$$\parallel f_i(a) - f_i(b) \parallel^2 \leq L \parallel a-b \parallel^2 \qquad (3)$$

or $\exists$ a non-negative C, the following inequality holds.

- *Y. Zhao, Y. Ming, and J. Yin are with the College of Computer, National University of Defense Technology, Changsha 410073, P.R. China. E-mail: {zhaoyawei09, jpyin}@nudt.edu.cn, and myw123@163.com.*
- *D. Guo is with the Science and Technology on Information System Engineering Laboratory, National University of Defense Technology, Changsha 410073, P.R. China. E-mail: guodeke@gmail.com.*

$$\parallel \nabla f_i(a) - \nabla f_i(b) \parallel \leq C \qquad (4)$$

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

$$F(a) \geq F(b) + \nabla F(b)^{\mathrm{T}}(a-b) + \frac{\gamma}{2} \parallel a-b \parallel^2 \qquad (5)$$

.

The inequality can be presented equivalently as follows.

$$\parallel F(a) - F(b) \parallel \geq \gamma \parallel a-b \parallel \qquad (6)$$

.

Many machine learning algorithms can be represented by this abstraction including regression and classification and so forth.

Inspired by the variance reduction in [**?**], we present the overview of our asynchronous and distributed SGD in Algorithm **??**. Our asynchronous and distributed SGD consists of three ingredients: epochs of iterations (the outer for loop at Line 2), random update strategy of parameters (the inner for loop at Line 6) and update rule of parameters (Line 8 and 9). We abstract all the threads running on machines of a cluster as two ingredients: the servers and the trainers. The epochs of iterations is conducted by the servers which adopts asynchronous communication protocol, i.e., SSP to proceed the iterations for all trainers. The random update strategy of parameters is conducted by the trainers which randomly pick one of n candidate training samples, compute the variance reduced gradient, and then update such gradient to parameters.

Algorithm 1 DisSVRG

1: Initialize $\tilde{\omega}^0$
2: **for** $s = 1, 2, ...$ **do** % Asynchronous learning parameters for all trainers
3:      $\tilde{\omega} = \tilde{\omega}^{s-1}$
4:      $\mu = \frac{1}{n} \sum\limits_{i=1}^{n} \nabla f_i(w_i)$
5:      $\omega_0 = \tilde{\omega}$
6:      **for** $t = 1, 2, ...m$ **do** % Variance reduction technique used in each trainer
7:          randomly pick $i_t \in \{0, 1, 2, ..., m\}$
8:          $v_t = \nabla f_{i_t}(w_{t-1}) - \nabla f_{i_t}(\tilde{\omega}) + \mu$ % stochastic variance reduced gradient
9:          $w_t = w_{t-1} - \eta v_t$
10:      **end for**
11:      $\tilde{w}_s = w_m$
12: **end for**

A global state of parameters is maintained by all servers. When the iteration begins, all trainers pull the global parameters from one of servers. Then all the trainers start to conduct the machine learning tasks asynchronously. To be specific, each a trainer computes the variance reduced gradients and update its local copy of global parameters. When a trainer finish to update its local parameters, it will push its local copy of parameters to one of servers. Since the runtime environment of machines are various, the computation time of variance reduced gradient is not same. However, the fast trainers will continue to proceed the iterations, and not wait the slow ones. On the other hand, the asynchronous SGD may not converge if all the trainers update parameters in a fully asynchronous way. Here, we set a delay bounded, i.e., τ. The delay τ is used to synchronize all the trainers. Specifically, when the fastest trainer finishes the tth iteration, and the slowest trainer does not finish the $(t - \tau)$th iteration, it will be forced to stop and wait the slowest trainer.

4 SYSTEM IMPLEMENTATION

DisSVRG is implemented by the servers and trainers where the servers maintain the local state of parameters, and the trainers compute the updates of parameters. The servers receive the updates of parameters from trainers, and then aggregate these updates into the local parameters. Each a trainer pushes a copy of a global parameters before it starts to conduct machine learning tasks. When a trainer finishes the updates of parameters during an epoch, it will pull the latest local parameters to one of servers. If a trainer has finished the tth epochs, and all the trainers have finished the $t - \tau$th epochs, then this trainer will pull the aggregated parameters and starts to another epoch, or it will have to wait. To be specific, the details of DisSVRG are presented as follows.

     Server:
     Trainer:

5 OPTIMIZATION OF ASYNCHRONOUS AND DISTRIBUTED SGD

In this section, we present two optimization tricks which can be used to accelerate DisSVRG in a cluster.

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** other trainers have finished $e_p - \tau$th epoch **then**
6:              send a copy of the global parameters.
7:          **end if**
8:      **end if**
9:      **if** receive a push request from the trainer p **then**
10:          receive the parameters from the trainer p, and aggregate them with the local parameters.
11:      **end if**
12: **end while**

Algorithm 3 Trainer

1: $e_p = 0.$
2: **while** true **do**
3:      send a pull request to a server
4:      **while** true **do**
5:          **if** receive a copy of global parameters from the server **then**
6:              $e_p = e_p + 1.$
7:              **for** $t = 0, 1, 2, ..., m$ **do**
8:                  random pick a non-negative number i with $i \in \{0, 1, .2, ..., n\}$
9:                  $v_t = \nabla f_{i_t}(w_{t-1}) - \nabla f_{i_t}(\tilde{\omega}) + \mu$
10:                  $w_t = w_{t-1} - \eta v_t$
11:              **end for**
12:              send the latest parameter w_m to a server.
13:          **end if**
14:      **end while**
15: **end while**

5.1 Learning rate with 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 inner properties can be exploited to accelerate the convergence of the machine learning algorithms.

Based on the inequality **??**, when b is valued the optimal weight, i.e., w^*, $\forall w$, it holds that $\| \nabla f_i(w) \| \leq C$. That is, when the parameters are updated, the absolute value of the gradient is bounded by a constant. Besides, the gradient of parameters decreases when such parameters approach to the optimal state. Especially, when the parameters are close to the optimal state, their gradient reaches zero approximately. Therefore, we introduce an accelerated factor σ with $\sigma = \| \nabla f_i(w) \|$. The learning rate of our SGD contains two ingredients: the constant and the accelerated factor. The constant part of DisSGD prevalents it to get struck at the local optimal state; while the accelerated factor makes DisSGD converge at a more than constant rate. Furthermore, since the gradient of each function $f_i(w)$ has an upper bound, the accelerated factor do not become very large and lead to the divergence of the objection.

5.2 Adaptive strategy of random updates with window of loss function

6 DISCUSSION

The machine learning tasks such as Deep learning are generally fed with extremely large volume of training data in the era of Big Data. However, conventional serial variants of SGD cannot consume training data on a single machine within the available time. Although distributed variants of SGD such as PetuumSGD are designed to solve this challenge, those existing SGD versions are always presented as demos of some distributed machine learning systems like Petuum. SGD has its inner weakness such as the variance due to the random update of parameters. Such those distributed machine learning systems thus decrease the learning rate to reduce the variance, which leads to slow convergence of SGD. The main reason is that when machine learning tasks are proceeded, the learning rate will be decreased to a very small value so that every iteration bring tiny benefits to the loss function. Even though existing distributed machine learning systems have made much optimization on the communication mechanism, SGDs embedded on those systems are not exploited to accelerate machine learning tasks.

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 distributed SGD adopts SSP as the consistency protocol which is suitable to iterative convergence machine learning algorithms. The most difference between our implementation of distributed SGD between PetummSGD is the variance technique adopted in the algorithm. PetuumSGD adopts standard SGD with the decreasing learning rate $O(\frac{1}{\sqrt{T}})$. Here, T means the epochs of iterations. Variance of PetuumSGD will be decreased when the learning rate becomes small. However, our implementation of SGD has been accelerated with two ingredients: the constant factor and the accelerated factor. The constant factor makes sure that our SGD keep a constant rate to reach to the optimal value. The accelerated factor will make the machine learning algorithm converge fast. Even though our SGD adopts the same consistency protocol with PetuumSGD, our SGD trains parameters at a faster rate than PetuumSGD by using a powerful variance technique.

Comparing with SVRG, our SGD is a asynchronous and distributed version. In the era of Big Data, the size of training data is extremely huge so that a single machine cannot handle all the training data. Since the service of data center is trivial to use, it is natural to perform machine learning algorithms with a cluster, instead of an machine. Inspired by the variance reduction technique of SVRG, we implement a new distributed SGD. Our SGD 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 accelerating factor which exploits the potential benefits of the loss function and make the machine learning algorithms converge fast. Third, the variance reduction of SVRG needs at most m random updates of parameters. m is valued as the times of the size of training data, which is not practical for a large volume of training data. Our SGD achieve the value of m by maintaining a window of the value of loss function and then analyzing them dynamically. Such an identification of m is suitable to the practical scenarios because that each learning thread can have its own setting of m. The reason is that the machines in the cluster may be very heterogeneous, and the learning threads are not synchronized during an epoch of iteration. The adaptive mechianism of the value of m is suitable to such situations.

Additionally, comparing with the SGD in [?], our SGD adopts a more natural way to implement distributed SGD with the variance reduction. SGD in [?] implements τ-delay bounded consistency protocol in the inner loop of SVRG; while the epochs of iterations keep fully consistency protocol, namely, BSP. Therefore, SGD in [?] has at least two weakness. First, the BSP consistency protocol for the epochs of iterations is not suitable to the iterative convergence machine learning tasks. Since the straggler problem exists in the learning threads due to the variety of system runtime environment, the fast learning threads in SGD in [?] have to wait the slow learning ones in an epoch. Considering that machine learning algorithms are iteratively convergent, such BSP consistency protocol wastes too much time. Second, the SGD in [?] is coupled with the hardware of clusters. The reason is that SGD in [?] uses m learning threads to perform machine learning tasks where m is the number of inner loop in an epoch. m in SGD in [?] is valued $O(n)$ where n represents the size of training data. Considering the huge size of training data, m is really large, which is not practical to a cluster. Instead, m in our SGD is identified by the adaptive and dynamical mechanism, which is flexible to adjust during the calculation of parameters.

7 PERFORMANCE EVALUATION

8 CONCLUSION