Journal of Parallel and Distributed Computing xxx (xxxx) xxx

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

WP-SGD: Weighted parallel SGD for distributed unbalanced-workload training system

Cheng Daning [a,b], Li Shigang [a,c,*], Zhang Yunquan [a]

[a] SKL of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, China
[b] University of Chinese Academy of Sciences, China
[c] Department of Computer Science, ETH Zurich, Switzerland

A R T I C L E I N F O

Article history:
Received 27 June 2017
Received in revised form 8 April 2020
Accepted 14 June 2020
Available online xxxx

Keywords:
SGD
Unbalanced workload
SimuParallel SGD
Distributed system

A B S T R A C T

Stochastic gradient descent (SGD) is a popular stochastic optimization method in machine learning. Traditional parallel SGD algorithms, e.g., SimuParallel SGD (Zinkevich, 2010), often require all nodes to have the same performance or to consume equal quantities of data. However, these requirements are difficult to satisfy when the parallel SGD algorithms run in a heterogeneous computing environment; low-performance nodes will exert a negative influence on the final result. In this paper, we propose an algorithm called weighted parallel SGD (WP-SGD). WP-SGD combines weighted model parameters from different nodes in the system to produce the final output. WP-SGD makes use of the reduction in standard deviation to compensate for the loss from the inconsistency in performance of nodes in the cluster, which means that WP-SGD does not require that all nodes consume equal quantities of data. We also propose the methods of running two other parallel SGD algorithms combined with WP-SGD in a heterogeneous environment. The experimental results show that WP-SGD significantly outperforms the traditional parallel SGD algorithms on distributed training systems with an unbalanced workload.

1. Introduction

The training process in machine learning can essentially be treated as the solving of the stochastic optimization problem. The objective functions are the mathematical expectation of loss functions, which contain a random variable. The random variables satisfy a known distribution. The machine learning training process can be formalized as

$$\min E[g(X,w)](X \sim \text{certain distribution } D) = \min \int_{\Omega} g(x,w)density(x)\Delta x \quad (1)$$

where $g(\cdot)$ is the loss function, w is the variables, X is the random variable, and $density(\cdot)$ is the probability density function of the distribution D.

Because some distributions cannot be presented in the form of a formula, we use the frequency to approximate the product of probability density $density(x)$ and Δx, as a frequency histogram can roughly estimate the curve of a probability density function. Thus, for a dataset, the above formula can be written in the following form:

$$\min E[g(X,w)](X \sim \text{certain distribution } D) \approx \min \frac{1}{m}\sum_{i=1}^{m} g(x^i, w) \quad (2)$$

where m is the number of samples in the dataset, and x^i is the ith sample value.

Stochastic gradient descent (SGD) is designed for the following minimization problem:

$$\min c(w) = \frac{1}{m}\sum_{i=1}^{m} c^i(w) \quad (3)$$

where m is the number of samples in the dataset, and $c^i : \ell_2 \mapsto [0,\infty]$ is a convex loss function indexed by i with the model parameters $w \in \mathbb{R}^d$. Normally, in the case of regularized risk minimization, $c^i(w)$ is represented by the following formula:

$$c^i(w) = \frac{\lambda}{2}\|w\|^2 + L(x^i, y^i, w^T x^i) \quad (4)$$

where $L(\cdot)$ is a convex function in $w \cdot x$. It is of note that in the analysis and proof, we treat model parameters, i.e., w, as the random variable during the training process.

When $L(x^i, y^i, w \cdot x^i)$ is not a strongly convex function, for example a hinge loss, the regularized term would usually guarantee the strongly convexity for $c^i(w)$.

* Corresponding author at: Department of Computer Science, ETH Zurich, Switzerland.
E-mail addresses: chengdaning@ict.ac.cn (D. Cheng), shigangli.cs@gmail.com (S. Li), zyq@ict.ac.cn (Y. Zhang).

https://doi.org/10.1016/j.jpdc.2020.06.011
0743-7315/© 2020 Elsevier Inc. All rights reserved.

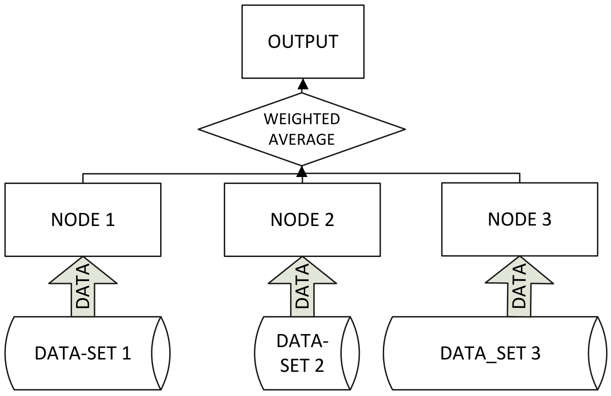

Fig. 1. Working pattern of WP-SGD when the quantities of data differ.

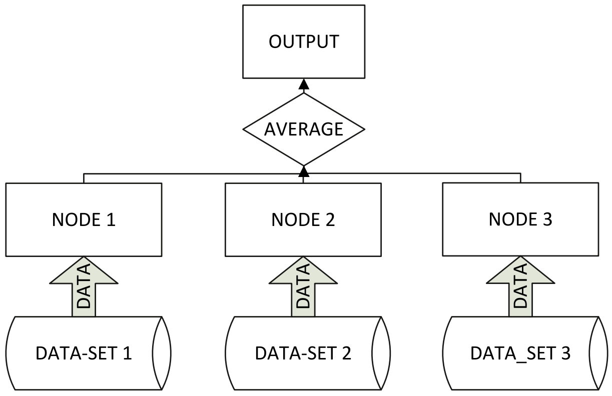

Fig. 2. Working pattern of SimuParallel SGD with equal quantities of data.

The iteration step for sequential SGD is

$$w_n = w_{n-1} - \eta \partial_w c^i(w_{n-1}) \tag{5}$$

Because of its ability to solve machine learning training problems, its small memory footprint, and its robustness against noise, SGD is currently one of the most popular topics [3,6,8,10,17–19].

As SGD was increasingly run in parallel computing environments [1,13], parallel SGD algorithms were developed [11,29]. However, heterogeneous parallel computing devices, such as GPUs and CPUs or different types of CPU, have different performance. The cluster may contain nodes having different computing performance. At the same time, parallel SGD algorithms suffer from performance inconsistency among the nodes [11]. Therefore, it is necessary to tolerate a higher error rate or to use more time when running parallel SGD algorithms on an unbalanced-workload system.

```
Input: Examples $\{c^1, \dots, c^m\}$, learning rate $\eta$, $k$ nodes in
       system;
Output: $v$
1  Randomly partition the examples;
2  for all $i \in \{1, \dots, k\}$ parallel do
3      Randomly shuffle the data on machine $i$;
4      Initialize $w_{i,0} = 0$;
5      Define the fastest nodes consuming $t$ samples;
6      Define the delay between the fastest node and the $i$th
       node as $T_i$;
7      for all $n \in \{1, \dots, t - T_i\}$ do
8          Get the $n$th example on the $i$th node, $c^{i,n}$;
9          $w_{i,n} = w_{i,n-1} - \eta \partial_w c^{i,n}(w_{i,n-1})$;
10     end
11     Based on training process, setting parameter $r$;
12     Compute $q_{r,i}$:
           $q_{r,i} = \frac{r^{T_i}}{\sum_{j=1}^{k} r^{T_j}}$        (6)
13 end
14 Aggregate from all nodes $v = \sum_{i=1}^{k} q_{r,i} \cdot w_{i,t}$;
15 Return $v$;
```

Algorithm 1: WP-SGD

In this paper, we propose the following weighted parallel SGD (WP-SGD) for a distributed training system with an unbalanced workload. WP-SGD is given as Algorithm 1. WP-SGD adjusts the weights of model parameters from each node according to the quantity of data consumed by that node. The working pattern of WP-SGD is illustrated in Fig. 1.

In WP-SGD, $r \leq (1 - \lambda\eta)$ and r is a parameter which is decided during the training process,[1] η is the learning rate and λ is the regularization parameter. We determine the value of the r via experience, data fitting, or analysis of training data and $L(\cdot)$. When $L(\cdot)$ is not a strongly convex function, r is close to the $1 - \lambda\eta$.

WP-SGD is based on SimuParallel SGD [29], which is shown as Algorithm 2. The working pattern of SimuParallel SGD is illustrated in Fig. 2.

```
Input: Examples $\{c^1, \dots, c^m\}$, learning rate $\eta$, $k$ nodes in
       system;
Output: $v$
1  Randomly partition the examples;
2  for all $i \in \{1, \dots, k\}$ parallel do
3      Randomly shuffle the data on machine $i$;
4      Initialize $w_{i,0} = 0$;
5      All nodes consume $t$ samples;
6      for all $n \in \{1, \dots, t\}$ do
7          Get the $n$th example on the $i$th node, $c^{i,n}$;
8          $w_{i,n} = w_{i,n-1} - \eta \partial_w c^{i,n}(w_{i,n-1})$;
9      end
10 end
11 Aggregate from all nodes $v = \sum_{i=1}^{k} \frac{1}{k} \cdot w_{i,t}$;
12 Return $v$;
```

Algorithm 2: SimuParallel SGD

The main bottleneck for SimuParallel SGD in the heterogeneous parallel computing environment is that we need to guarantee that all nodes have trained on equal quantities of data before we average them (Line 5 and Line 11, respectively, in Algorithm 2). This requirement leads to a degradation in performance on the heterogeneous cluster. WP-SGD uses a weighted average operation to break this bottleneck. WP-SGD does not require all nodes to be trained on equal quantities of data and incorporates the delay information into the weights (Line 5, Line 6, and Line 12 with Eq. 6), which allows WP-SGD to run efficiently in a heterogeneous parallel computing environment.

WP-SGD suggests that when the workload is unbalanced within the cluster and there is a delay between the fastest node and the ith node, the weight of the model parameters on the ith node should be decreased exponentially.

[1] r is the contracting map rate for the SGD framework, which we will give its explanation in following parts.

The upper bound of the objective function value calculated by the output of WP-SGD will be less than the upper bound of the objective function value of the output from sequential SGD in the fastest node, when solving problems with more variance (smaller regularization constant) [29], and the system parameters satisfy Eq. (7).

$$2\sum_{i=1}^{k} r^{T_i} > \sqrt{k} + k \tag{7}$$

where k is the number of nodes in this system.

What is more, WP-SGD is able to combine traditional synchronous parallel SGD algorithm, i.e., mini-batch SGD, and asynchronous parallel SGD algorithm, i.e., delay SGD.

A numerical experiment on data from KDD Cup 2010 [26] and Real-sim dataset show the following conclusion:

1. The final output of WP-SGD with an unbalanced workload can be nearly equivalent to the output from a system with a perfectly balanced workload with the best performance nodes. In a workload-unbalanced environment, WP-SGD uses less time than workload-balanced SGD.

2. For inappropriate algorithms or synchronous algorithms on the unbalanced workload system, parallel technology brings negative effects.

3. Combining with WP-SGD, traditional synchronous algorithms overcome the system's workload unbalanced problems.

The key contributions of this paper are as follows:

1. We propose a novel parallel SGD algorithm, WP-SGD, for a distributed training system with unbalanced workloads.

2. We prove that WP-SGD can tolerate a large delay between different nodes. WP-SGD suggests that when there is an increase in the delay between the fastest node and the ith node, the weight of the model parameters for the ith node should be decreased exponentially.

3. We provide the results of experiments which we conducted using real-world data to demonstrate the advantages of WP-SGD on computing environment with unbalanced workloads.

In the next section, we present related works. In Section 3 we present a basic view of traditional parallel SGD algorithms. In Section 4, we demonstrate the analysis and proof for WP-SGD. In Section 5, we theoretically offer some complementary technologies based on WP-SGD. In Section 6, we present the results of the numerical experiments.

2. Related work

SGD dates back to early work by Robbins and Monro et al. [2,17]. In recent years, combined with the GPU [1,13], parallel SGD algorithms have become one of the most powerful weapon for solving machine learning training problems [8,9,14]. Parallel SGD algorithms can be roughly classified into two categories, which we call delay SGD algorithms and model average SGD algorithms.

Delay SGD algorithms first appeared in Langford et al.'s work [14]. In a delay SGD algorithm, current model parameters add the gradient of older model parameters in τ iterations (τ is a random number where $\tau < \tau_{max}$, in which τ_{max} is a constant). The iteration step for delay SGD algorithms is

$$w_n = w_{n-1} - \eta \partial_w c^i(w_{n-\tau}) \tag{8}$$

In the Hogwild! algorithm [11], under some restrictions, parallel SGD can be implemented in a lock-free style, which is robust to noise [4]. However, these methods lead to the consequence that the convergence speed will be decreased by $o(\tau^2)$. To ensure the delay is limited, communication overhead is unavoidable, which hurts performance. The trade-off in delay SGD is between delay, degree of parallelism, and system efficiency:

1. Low-lag SGD algorithms use fewer iteration steps to reach the minimum of the objective function. However, these algorithms limit the number of workers and require a barrier, which is a burden when engineering the system.

2. Lock-free method is efficient for engineering the system, but the convergence speed, which depends on the maximum lag, i.e. τ in Eq. (8), is slow.

3. The lower limit of the delay is the maximum number of workers the system can have.

From the point of view of engineering implementation, the implementation of delay SGD algorithms is accomplished with a parameter server. Popular parameter server frameworks include ps-lite in MXNet [5], TensorFlow [1], and Petuum [24]. A method that constricts the delay was offered by Ho et al. [12]. However, if the workers in the parameter server have different performance, τ is increased, causing convergence speed to be reduced.

Delay SGD algorithms can be considered as an accelerated version of sequential SGD. Model average SGD algorithms accelerate SGD via the averaging of model parameters. Zinkevich et al. [29] proposed SimuParallel SGD, which has almost no communication overhead. Y. Zhang et al. [28] gave an insightful analysis and proof for this parallel algorithm. However, these methods do not take into account the heterogeneous computing environment. J. Zhang et al. [28] also point out the invalidity of SimuParallel SGD. In fact, the effect of a model average SGD depends primarily on how large the model parameters' relative standard deviation is, which means it is a trade-off between the parallelism and the applicability for dataset.

From the point of view of engineering implementation, model average SGD algorithms can be implemented in a MapReduce manner [7]. Thus, most of them are running on platforms like Hadoop [22]. If the nodes in the cluster have different performance, the slowest node is the performance bottleneck.

In HPC & AI area, model average SGD algorithm and its variant, batch SGD and decentralized SGD, are the most important parallel and distributed SGD algorithm [15,21,23,25]. However, those algorithms often require all nodes to have the same performance or to consume equal quantities of data. Thus, unbalanced workload is the key performance bottleneck for model average SGD algorithm on HPC. What is more, with the development of heterogeneous parallel computing devices, heterogeneous parallel computing cluster is the main stream high performance computing cluster style. The cluster may contain nodes having different computing performance. Heterogeneous parallel computing cluster exacerbates unbalanced workload problem.

Above parallel SGD algorithms have various superb features. However, all of them lack robustness against an unbalanced workload.

3. Background and main idea

3.1. Notation and definitions

Notation	Definition
t	The number of samples consumed by the fastest nodes
T_i	The delay between the fastest nodes and the ith node.Here we define the delay as the gap of the number of data samples between the fast nodes and node i.
x^j	The jth sample value
y^j	The label for the jth sample
λ	The parameter for the regularization term. For some loss functions, such as hinge loss, it guarantees strongly convexity.
η	Step length or learning rate for SGD
w	Variables for objective function. In machine learning, it is the model parameters.
X	The random variable
m	The number of samples in the dataset
$c(\cdot)$	Loss function
$q_{r,i}$	In WP-SGD, the weight for the ith node on contracting map rate r
$L(x^j, y^j, w \cdot x^j)$	The loss function without a regularization term
D	The distribution for the random variables
k	The total number of nodes in a cluster
r	The contracting map rate for $c(w)$ in SGD
τ	In delay SGD, the delay between the current model parameters and the older model parameters when current model updates itself by the information of the older model parameters.
τ_{max}	Maximum number of τ
D^*_η	The distribution of the unique fixed point in SimuParallel SGD and WP-SGD, with learning rate η
D^t_η	The distribution of the stochastic gradient descent update after t updates, i.e., $M^{i,t-T_i}$, with learning rate η.
$M^{i,t-T_i}$	A random variable whose PDF is D^t_η. The output of the ith node after $t - T_i$ iterations.
$M^{\#,t}$	A random variable. The output of WP-SGD, where the fastest node trained on t samples
$D^{\#,t}_\eta$	The distribution of $M^{\#,t}$
$W_z(X,Y)$	Wasserstein distance between two distributions X, Y
s	In more average operation SimuParallel SGD and WP-SGD, which are offered at Section 5, the span between two average operations from the view of the fastest nodes
v	The final output of an algorithm
μ_X	The mean of the random variable X
σ_X	The standard deviation of the random variable X
$d(\cdot,\cdot)$	Euclidean distance.

We collect our common notations and definitions in this subsection.

Definition 1 (*Lipschitz Continuity*). A function $f \colon \mathcal{X} \mapsto \mathbb{R}$ is Lipschitz continuous with constant C with respect to a distance, in this paper, we use Euclidean distance, d if $|f(x) - f(y)| \le Cd(x,y)$ for all $x, y \in \mathcal{X}$.

Definition 2 (*Lipschitz Seminorm*). Luxburg and Bousquet [16] introduced a seminorm. With minor modification, we use

$$\|f\|_{\text{Lip}} = \inf\{C : |f(x) - f(y)| \le Cd(x,y) \text{ for all } x, y \in \mathcal{X}\} \quad (9)$$

That is, $\|f\|_{\text{Lip}}$ is the smallest constant for which Lipschitz continuity holds.

In the following, we let $\|L(x,y,y')\|_{\text{Lip}} \le G$ as a function of y' for all occurring data $(x,y) \in \mathcal{X} \times \mathcal{Y}$ and for all values of w within a suitably chosen (often compact) domain. G is a constant.

Definition 3 (*Contracting Map*). $d(\cdot,\cdot)$ is Euclidean distance, for a map f, and exists a point x^*, if

$$d(f(x), x^*) < rd(x, x^*) \quad (10)$$

holds for all x in domain and $r < 1$. We name f as contracting map, x^* is the fixed point and r is contracting map rate.

Definition 4 (*Relative Standard Deviation of X with respect to a*).

$$\sigma^a_X = \sqrt{E(X-a)^2} \quad (11)$$

As we can see, $\sigma_X = \sigma^{\mu_X}_X$, where μ_X is the mean of X.

Supertabular 3.1 shows the notations used in this paper and the corresponding definitions.

3.2. Introduction to SGD theory

The core proof idea of SGD's proof in paper [29] is that with the process of SGD, the PDF of model parameters will converge into a fixed PDF. Thus, all discussions about the change in the training process are about random variables. When we draw a specific realization about model parameters , i.e., a specific machine learning model at t iteration, from model parameters' PDF, it is an independent process.

Theorem 1, Theorem 2, Theorem 3, and Lemma 1 are key theorems we will use. All four theorems are proved by Zinkevich et al. [29].

Theorem 1 ([29, Theorem 11]). *Given a cost function c that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, and a distribution D such that σ_D is bounded, then for any point p*

$$\begin{aligned} &E_{p \in D}[c(p)] - \min_w c(w) \le \\ &\sigma^p_D \sqrt{2\|\nabla c\|_{\text{Lip}}(c(p) - \min_w(w))} \\ &+ (\|\nabla c\|_{\text{Lip}}(\sigma^p_D)^2/2) + (c(p) - \min_w c(w)) \end{aligned} \quad (12)$$

Theorem 1 highlights the relationship between the distribution of model parameters and $\min_{w \in \mathbb{R}^d} c(w)$, which is the expected result of SGD when p is equal to w.

Theorem 2 ([29, Theorem 9]).

$$c(E_{w \in D^*_\eta}[w]) - \min_{w \in \mathbb{R}^d} c(w) \le 2\eta G^2 \quad (13)$$

where D^*_η is the distribution of a unique fixed point in SimuParallel SGD. This theorem provides an idea of the bound on the third part of Theorem 1.

Lemma 1 ([29, Lemma 30]).

$$\sigma^c_X \le \sigma^{c'}_X + d(c, c') \quad (14)$$

where $d(\cdot,\cdot)$ is the Euclidean distance.

Theorem 3 *([29, Theorem 70]). If D_η^t is the distribution of the stochastic gradient descent update after t iterations, then*

$$d(\mu_{D_\eta^t}, \mu_{D_\eta^*}) \le \frac{G}{\lambda}(1-\eta\lambda)^t \tag{15}$$

$$\sigma_{D_\eta^t} \le \frac{2\sqrt{\eta}G}{\sqrt{\lambda}} + \frac{G}{\lambda}(1-\eta\lambda)^t \tag{16}$$

The above theorems describe how and why SGD can converge to a minimum. The difference between the value of $c(\cdot)$ using the output w from SGD and the minimum of $c(\cdot)$ is controlled by three factors:

(1) The difference between the expectation of the current distribution of model parameters and the expectation of D_η^*

(2) The standard deviation of the distribution of the current model parameters, which is $\sigma_{D_\eta^t}$

(3) The difference between the expected value of $c(w)$ when w satisfies distribution D_η^* and the minimum value of $c(\cdot)$

For the sequential SGD, carrying out the algorithm would reduce the first part and the second part. The third part is controlled by η and $L(\cdot)$.

For SimuParallel SGD, the first part and the third part are the same for different nodes. However, $\sigma_{D_\eta^t}$ can benefit from the averaging operation. SimuParallel SGD uses the gain in the standard deviation to reduce the number of iteration steps needed to reduce the first and second parts. In other words, SimuParallel SGD accelerates SGD.

4. Proof and analysis

4.1. Analysis of WP-SGD

The concept of WP-SGD has two main aspects:

1. Our proposed weight is to compensate for the main loss from the delay between the different nodes. The main loss from the delay is controlled by the exponential term $(1-\lambda\eta)^t$.

2. Under the condition that the gain from the standard deviation's reduction is greater than the loss in the mean's weighted average from the perspective of the fastest node, the WP-SGD output will outperform the fastest node.

All of the following lemmas, corollaries, and theorems are our contributions.

We focus on the first aspect at the beginning: Corollary 1 and Lemma 3 show how the mean and standard deviation will change by using WP-SGD. Their sum is the upper bound of the relative standard deviation which is shown in Lemma 1.

Lemma 2 is used in the proof of Corollary 1.

Lemma 2. *Suppose that $X^1 \dots X^k, B$ are independent distributed random variables over $\mathbb{R}^d$. Then if $A = \sum_{i=1}^k q_i \cdot X^i$ and $1 = \sum_{i=1}^k q_i$, it is the case that*

$$d(\mu_A, \mu_B) \le \sum_{i=1}^k q_i \cdot d(\mu_{X^i}, \mu_B)$$

In following part of section 4.1, we give theory based on $q_{1-\lambda\eta,i}$ setting.

Corollary 1. *The fastest node consumes t data samples. $D_\eta^{t-T_i}$ is the distribution of model parameters updated after $t - T_i$ iterations in node i, and $D_\eta^{\#,t}$ is the distribution of the stochastic gradient descent update in WP-SGD.*

$$d(\eta_{D_\eta^{\#,t}}, \eta_{D_\eta^*}) \le \frac{Gk(1-\eta\lambda)^t}{\lambda\sum_{i=1}^k(1-\eta\lambda)^{T_i}} \tag{17}$$

Lemma 3. *$M^{i,t-T_i}$ is the output of node i. Then, if*

$$M^{\#,t} = \sum_{i=1}^k q_{1-\eta\lambda,i} \cdot M^{i,t-T_i} \tag{18}$$

then the distribution of $M^{\#,t}$ is $D_\eta^{\#,t}$. It is the case that

$$\sigma_{D_\eta{}^{\#,t}} \le \frac{\sqrt{k}}{\sum_{i=1}^k(1-\eta\lambda)^{T_i}}\left(\frac{2G\sqrt{\eta}}{\sqrt{\lambda}} + \frac{G}{\lambda}(1-\eta\lambda)^t\right) \tag{19}$$

Combining Lemma 1, Theorem 1, Corollary 1 whose proof uses Lemma 2, and Lemma 3, we have the following:

Theorem 4. *Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, the bound of WP-SGD is*

$$E_{w\in D}[c(w)] - \min_w c(w) \le \left(\left(\frac{Gk(1-\lambda\eta)^t}{\lambda\sum_{i=1}^k(1-\lambda\eta)^{T_i}} + \frac{\sqrt{k}}{\sum_{i=1}^k(1-\lambda\eta)^{T_i}}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}(1-\lambda\eta)^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}} + \sqrt{2\eta G^2}\right)^2 \tag{20}$$

Next, we discuss the second aspect.

It is apparent that there is no guarantee that the output of WP-SGD will be better than the output from the fastest nodes, because from the viewpoint of the best-performing node, the weighted average will damage its gain from contraction of the mean value term. Here, we offer Corollary 2 that defines the conditions under which the output from the fastest nodes will benefit from the normal-performance nodes. In the following, $W_z(X, Y)$ is the Wasserstein distance between two distributions X, Y, and the fastest nodes consume t data samples in an unbalanced-workload system.

Corollary 2. *For WP-SGD, when*

$$\frac{\sum_{i=1}^k(1-\eta\lambda)^{T_i} - \sqrt{k}}{k - \sum_{i=1}^k(1-\eta\lambda)^{T_i}} > \frac{(1-\eta\lambda)^t W_1(D_\eta^{\#,1}, D_\eta^*)}{(1-\eta\lambda)^t \cdot W_2(D_\eta^{\#,1}, D_\eta^*) + \sigma_{D_\eta^*}} \tag{21}$$

the upper bound of the objective function value of WP-SGD is closer to the minimum than is the upper bound of the objective function value of sequential SGD on the fastest nodes.

$W_z(D_\eta^{\#,1}, D_\eta^*)$ is not a prior value. However, Corollary 2 still eliminates the dataset whose $\sigma_{D_\eta^*}$ and $\sigma_{D_\eta^{\#,1}}$ are small. $\sigma_{D_\eta^{\#,1}}$ is positively related to the standard deviation of the dataset. The standard deviation of the dataset will influence the values of $W_1(D_\eta^{\#,1}, D_\eta^*)$ and $W_2(D_\eta^{\#,1}, D_\eta^*)$. In an extreme example, when all samples in the dataset are the same, i.e., SGD degenerates into Gradient Descent, i.e. GD, WP-SGD would be invalid, and this is also the case with SimuParallel SGD.

Most of the time, the standard deviations of real-world datasets, especially high dimension sparsity dataset, are usually large enough. In the case where the $\sigma_{D_\eta^{\#,1}}$ and $\sigma_{D_\eta^*}$ are large enough, under Corollary 3, WP-SGD would be better than the sequential SGD.

Corollary 3. *For WP-SGD, when*

$$2\sum_{i=1}^k(1-\eta\lambda)^{T_i} > \sqrt{k} + k \tag{22}$$

the upper bound of the objective function value of WP-SGD is closer to the minimum than is the upper bound of the objective function value of sequential SGD on the fastest nodes.

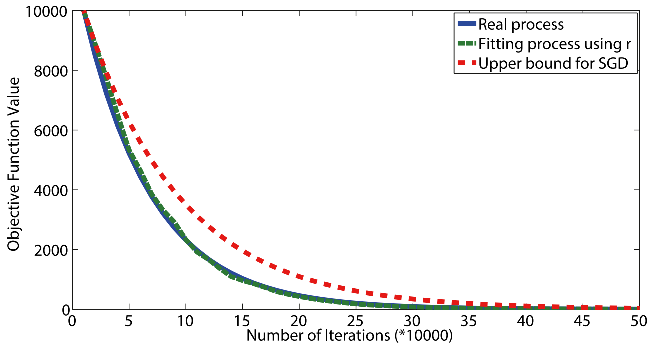

Fig. 3. An example of using contracting map rate r for fitting the actual contracting process. In this example, the objective function value decreases from 12 000 to zero in 500 000 iterations.

Corollary 3 suggests that WP-SGD can tolerate sufficient delay. As we can see, the robustness of whole system will be stronger as the scale of the cluster increases.

4.2. Analysis and redesign: weight for the dataset and $c(\cdot)$ whose contracting map rate is small

Above weight design using $1 - \lambda\eta$ is to meet the upper bound of SGD algorithm: The comparison between the upper bound shows the superiority between different algorithm. However, in practice, not all SGD processes which use different dataset and loss function reaches the upper bound.

For example, the convergence speed of log loss is faster than hinge loss; the convergence speed of the process which uses small variance dataset is faster than the convergence speed of the process which uses large variance dataset.

Thus, the weight of using $1 - \lambda\eta$ should be used in the condition where the equivalent condition of inequalities is achieved, i.e. $L(\cdot)$ is very close to being a linear function (the proof of Lemma 3 in Zinkevich et al.'s work [29]). This requirement means that $L(\cdot)$ is not a strongly convex function.

The nature of SGD is contracting map. Contracting map rate varies during the iteration process for every iteration in SGD process. When the loss function's second derivative is larger, or during the process, many of the samples' directions are parallel to the current model parameters' direction, the contracting map rate will be smaller. Therefore, from the standpoint of the overall iteration process rather than that of a single iteration, we should redesign a smaller contracting map rate to replace $(1 - \eta\lambda)$. We denote this new contracting map rate by r. Usually, r should be a smaller number when the direction of processing samples is closer to the direction of the current model parameters, i.e., w_n, and the second derivative of $L(\cdot)$ is larger.

We can determine the value of the new contracting map parameter via experience, data fitting, or analysis of training data and $L(\cdot)$, as in Fig. 3.

As we ascertain a value for the new contracting map rate r, we rewrite $q_{1-\lambda\eta,i}$ as $q_{r,i}$, Theorem 4 as Theorem 5 and Corollary 3 as Corollary 4:

$$q_{r,i} = \frac{r^{T_i}}{\sum_{j=1}^{k} r^{T_j}}$$

Theorem 5 (*Incorporating r into* Theorem 4). *Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, and in view of the overall process, the contracting map rate is r, and the bound of WP-SGD is*

$$E_{w\in D}[c(w)] - \min_{w} c(w)$$

$$\leq \left(\left(\frac{Gkr^t}{\lambda\sum_{j=1}^{k} r^{T_i}} + \frac{\sqrt{k}}{\sum_{j=1}^{k} r^{T_i}}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}r^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}} + \sqrt{2\eta G^2}\right)^2 \tag{23}$$

Corollary 4 (*Incorporating r into* Corollary 3). *Given that WP-SGD runs on a dataset having a large standard deviation and a large standard deviation of the fixed point, and in view of the overall process, the contracting map rate of $c(\cdot)$ is r, and when*

$$2\sum_{i=1}^{k} r^{T_i} > \sqrt{k} + k \tag{24}$$

the upper bound of the objective function value of WP-SGD is closer to the minimum than is the upper bound of the objective function value of sequential SGD on the fastest nodes.

5. Running popular parallel SGD algorithms combined with WP-SGD in heterogeneous environments

Current parallel SGD algorithms lack the feature of robustness in heterogeneous environments. However, they are characterized by a number of superb features such as the overlap between communication and computing (delay SGD) and fast convergence speed (model average SGD). It is reasonable to consider combining WP-SGD with these algorithms in order to gain the benefits of their excellent features and the adaptability to unbalanced-workload environments.

5.1. Combining WP-SGD with other model average SGD

Mini-batch SGD that averages the model parameters at each iteration [27] is the most important model average SGD. However, averaging at each iteration operation is expensive, and the mini-batch is more vulnerable to performance differences. There is a compromise parallel SGD algorithm that averages model parameters at a fixed s length. The number of *span* is from the point of the best performance nodes. Here we offer theoretical analyses of this parallel algorithm and its theoretical performance in unbalanced-workload environments, based on the analyses of WP-SGD.

Deduction 1. Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, we average parameters every s iterations for the fastest node in SimuParallel SGD. Then, the bound of the algorithm is

$$E_{w\in D}[c(w)] - \min_{w} c(w) \leq \tag{25}$$

$$\left(\left(\frac{Gr^t}{\lambda} + \frac{1}{\sqrt{k}^{t/s}}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}r^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}} + \sqrt{2\eta G^2}\right)^2 \tag{26}$$

Deduction 2. Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, we average parameters every s iterations for the fastest node in WP-SGD. Then, the bound of the algorithm is

$$E_{w\in D}[c(w)] - \min_{w} c(w) \leq \left(\left(\frac{Gr^t}{\lambda}\left(\frac{k}{\sum_{j=1}^{k} r^{T_i}}\right)^{t/s} + \left(\frac{\sqrt{k}}{\sum_{j=1}^{k} r^{T_i}}\right)^{t/s}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}r^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}} + \sqrt{2\eta G^2}\right)^2 \tag{27}$$

For all nodes with the same performance, the more average the operation, the closer the output model parameters will be to the function minimum. In this case, our consideration should be to balance the cost of operation and the gain from the “better” result. As is well known, not all training datasets' variances are large enough to get the expected effect. On an unbalanced-workload system, we should also guarantee that $\frac{\sqrt{k}}{\sum_{j=1}^{k} r^{T_j}} < 1$ to ensure overall that the training process is valid.

5.2. Combining WP-SGD with delay SGD

Because of the excellent adaptability on different kinds of datasets and the overlapping of the cost of communication and computing, delay SGD is widely used in machine learning frameworks such as MXNet [5], TensorFlow [1], and Petuum [24]. However, all of these algorithms are designed for a balanced-workload environment. In this section, we offer Algorithm 3, which combines WP-SGD and one kind of delay SGD to make delay SGD algorithms work efficiently in heterogeneous computing environments. Some intermediate variables are defined in the algorithm description. The working pattern of Algorithm 3 is illustrated in Fig. 4.

```
Input: Examples $\{c^1, \ldots, c^m\}$, learning rate $\eta$, $k$ nodes in system ;
Output: $v$
Randomly partition the examples;
Phase 1:
For Worker:
pull the $w_{i,j}$ from the $i$th Server;
calculate $\partial_w c_{i,j}(w_{i,j})$;
push the $\partial_w c_{i,j}(w_{i,j})$ to the Server;
For the $i$th Server
Initialize $w_{i,0} = 0$;
for $j \in (0 \ldots Forever)$ do
    receive $\partial_w c_{i,j-1-\tau}(w_{i,j-1-\tau})$ from the Worker;
    Initialize $Flag = true$;
    Call function $Check(w_{j-1-\tau} \cdots w_{j-1}, \lambda, \eta, x^j, Flag)$;
    if $Flag$ then
        $w_{i,j} := w_{i,j-1} - \eta \partial_w c^{i,j}(w_{i,j-1-\tau})$;
        Call function $Check(w_{j-2} \cdots w_j, \lambda, \eta, x^j, Flag)$;
    end
    if $!Flag$ then
        abandon $w_{i,j}$;
        j = j -1;
    end
end
Phase 2:
Aggregate $v$ from all Servers $v = \sum_{i=1}^{k} q_{r,i} \cdot w_{i,j}$;
Return $v$;
```

Algorithm 3: WP-SGD and delay SGD

The proof of Algorithm 3 focuses on two main key points: (1) to guarantee that all of $w_{n-\tau}$ to w_n is on one side of the fixed point in the direction of the sample, and (2) to determine the value of the maximum contraction map rate when using this kind of delay SGD. Both of above 2 key points are described in the proof of Lemma 4.

For the first point, when running the $(n+1)$th update step, we also need to ensure that the first n update steps satisfy the algorithm. The above requirement means that we should be able to find a range in which the projection of the unique fixed point in the current sample direction addressed. With the processing, the range should shrink. We calculate the range of the fixed point based on the latest iteration information at the beginning of each update step, like Fig. 5. We only accept the new model parameters that are on the same side of this range as the older model parameters; otherwise, we abandon these new model parameters and use another sample to recalculate new model parameters. The above operation is determined by the point of this range closest to the old model parameters (in Algorithm 3, this point is denoted ι_{min}). These processes are described in *Check* function in Algorithm 4.

For the second point, WP-SGD and Simul Parallel SGD share the same proof frame. In the proof of Simul Parallel SGD, the Lemma 3 in Zinkevich et al.'s work [29] decides the contracting map rate of Simul Parallel SGD. Here, we offer Lemma 4 for Algorithm 3. Using the proof frame of Simul Parallel SGD with following Lemma 4 instead of Lemma 3 in Zinkevich et al.'s work [29], we can find the contracting map rate of Algorithm 3 and finish the whole proof.

The details of Lemma 4's proof are offered in the Appendix.

```
Input: model parameters $\{w_{j-1-\tau}, \ldots, w_{j-1}\}$, regularization parameter $\lambda$,learning rate $\eta$, sample $x^j$, Output $Flag$;
for all $j_{tmp} \in \{j-1-\tau, \ldots, j\}$ do
    $\iota_{j_{tmp}} := x^j \cdot w_{j_{tmp}}$;
    $\iota_{j_{tmp}-1} := x^j \cdot w_{j_{tmp}-1}$;
    $\iota_{j_{tmp}-2} := x^j \cdot w_{j_{tmp}-2}$;
    $\beta^2 := x^j \cdot x^j$;
    $\iota_{j_{tmp}-1\perp} := w_{j_{tmp}-1} - \iota_{j_{tmp}-1}/\sqrt{\beta}$;
    $\iota_{j_{tmp}-2\perp} := w_{j_{tmp}-2} - \iota_{j_{tmp}-2}/\sqrt{\beta}$;
    $c^* := \left\| \frac{\partial L(y, \hat{y})}{\partial \hat{y}} \right\|$;
    $rate := \sqrt[\tau]{\lambda\eta + c^*\eta\beta^2}$;
    $\iota_{min} := \frac{\iota_{j_{tmp}-1} - rate \cdot \iota_{j_{tmp}-2}}{1 - rate}$;
    if $((\iota_{j_{tmp}} \notin [\iota_{min}, \iota_{j_{tmp}-1}]$ and $\iota_{j_{tmp}} \notin [\iota_{j_{tmp}-1}, \iota_{min}])$ or $\frac{\iota_{j-2\perp}}{\iota_{j-1\perp}} > 1)$ then
        $Flag = false$;
    end
end
```

Algorithm 4: *Check* function

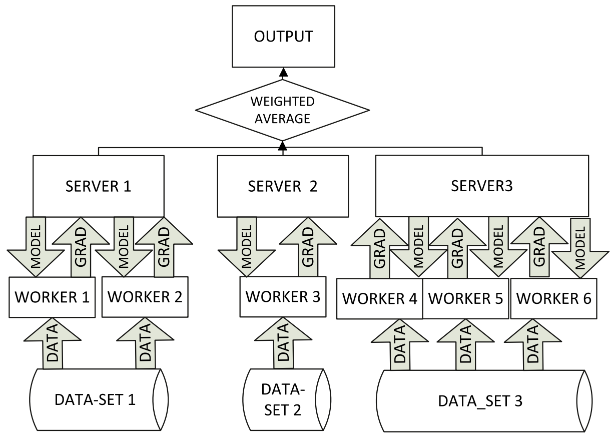

Fig. 4. Working pattern of Algorithm 3 when the quantities of data differ.

Lemma 4. *Let $c^* \geq \left\| \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \right\|$ be a Lipschitz bound on the loss gradient. Then if $\eta\lambda + \eta\beta_{max}^2 c^* \leq (1-\eta\lambda)^{\tau_{max}}$, the Algorithm 3 is a convergence to the fixed point in ℓ_2 with Lipschitz constant $1-\lambda\eta$. β_{max}^2 is defined as following formula: $\beta_{max}^2 = \max\|x^i\|^2$. τ_{max} is the maximum delay.*

As we discussed in Section 2, the maximum lag the system can tolerate is the maximum number of workers the system can have. When all workers have the same performance, the system will achieve the most efficient working state. In practice, it is very hard to let all nodes in an unbalanced-workload system have the same performance, especially when the clusters consist of different kinds of computing devices. Algorithm 3 is the algorithm designed for this kind of cluster.

5.3. Dalay & model average based WP-SGD

Combining above two algorithms, We propose a mixed parallel SGD algorithm, named as dalay & model average based WP-SGD, which is shown in algorithm 5.

```
Input: Examples $\{c^1, \dots, c^m\}$, learning rate $\eta$, $k$ nodes,
       allreduce span $s$;
Output: $v$
1  Randomly partition the examples;
2  for all $i \in \{1,\dots,k\}$ parallel do
3      Randomly shuffle the data on machine $i$;
4      Initialize $w_{i,0} = 0$;
5      Define the fastest nodes consuming $t$ samples;
6      Define the delay between the fastest node and the $i$th
       node as $T_i$;
7      Init conter $j_i = 0$;
8      for all $n \in \{1,\dots,t\}$ do
9          Algorithm 3 update $w$ one iteration;
10         if fastest nodes'$j_{fast}$ = $s$ then
11             Aggregate from all nodes $w = \sum_{i=1}^{k} q_{r,i} \cdot w_{i,t}$;
12             $j_i = 0$;
13         end
14         $j_i = j_i + 1$;
15     end
16 end
17 Aggregate from all nodes $v = \sum_{i=1}^{k} q_{r,i} \cdot w_{i,t}$;
18 Return $v$;
```

Algorithm 5: dalay & model average based WP-SGD

Based on the analysis of above algorithms, it is easy to gain the conclusion that dalay & model average based WP-SGD can converge into the w^*.

6. Numerical experiments

6.1. Platform

We conducted these experiments on a cluster consisting of different types of CPU nodes on Alibaba clouds. The detail information of the server in the cluster is shown in Table 1.

6.2. Algorithm and code setting

6.2.1. Delay & model average based WP-SGD

In the code of our experiment, one node uses four threads: three compute&bcast thread and one listening&synchronous

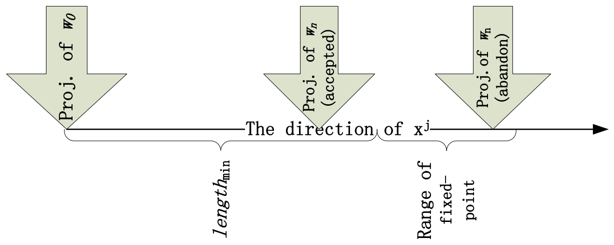

Fig. 5. Algorithm 3 only accepts the new model parameters that are on the same side of this range as the older model parameters (w_0 in this figure).

Table 1
The information of nodes in cluster.

# Nodes per Server	# Server	CPU	Net	Mem
1	1	Intel(R) Xeon(R) CPU E3-1225 v6 @ 3.30 GHz	1000 Mb/s	31 GB
2	3	Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.40 GHz	10000 Mb/s	116 GB
1	1	Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.40 GHz	10000 Mb/s	116 GB
2	4	Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20 GHz	10000 Mb/s	113 GB
2	2	Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80 GHz	1000 Mb/s	55 GB

thread. We use this setting for Intel(R) Xeon(R) CPU E3-1225 v6 server only have four cores.

In compute&bcast threads, three threads use parameter-server frame asynchronous parallel trains model: computing part consists of one server thread and two worker threads. The work of worker threads is pulling model, computing gradient, and pushing gradient. The work of server threads is receiving gradient, updating model, broadcasting the model.

In listening&synchronous thread, this thread is used to listen to the nodes which finished their workload and give the board cast signal to the server thread.

For we want to make results have good comparability in the different algorithms, we do not use OpenMP to accelerate worker threads. (As we can see, in E5-2640, E5-2660, E5-2680 nodes, we have free cores. If we use full resources, the performance of mini-batch SGD and SimulParallel SGD would be bad, because the performance of E5-2640 is 50% to 70% than other Server.)

In our experiments, we name this algorithm as D&M based WP-SGD.

6.2.2. WP-SGD

In our experiments, each node uses two threads as OpenMP threads to accelerate the process, because in delay&model average based WP-SGD algorithm, each node uses two worker threads to accelerate the computing process. We use this setting because we want all experimental results are comparability.

6.2.3. Simulparallel SGD

In our experiments, each node uses two threads as OpenMP threads to accelerate the process, which is the same as the configuration with WP-SGD.

6.2.4. Mini-batch SGD

In our experiments, each node uses two threads as OpenMP threads to accelerate the process, which is the same as the configuration with WP-SGD.

We use mini-batch SGD as benchmark because mini-batch SGD is the most widely used parallel SGD algorithm in HPC areas to train a machine learning model.

6.2.5. *Sequential SGD*

In our experiments, we only use one node on Intel(R) Xeon(R) CPU E3-1225 v6 node to measure the performance. We use this algorithm as a benchmark because, as a benchmark, other algorithms show the ability of parallel technologies.

In this experiment, the sequential SGD process uses two threads as OpenMP threads to accelerate the process, which is the same as the configuration with WP-SGD.

6.2.6. *Averaging the model parameters directly*

As the baseline, we used the nodes' output from WP-SGD nodes and the outputs created by using the direct averages of the model parameters. We name this algorithm as averaging directly.

We use this algorithm as a benchmark because we want to show that the error method to deal with workload unbalance problems may cause a catastrophe.

In our experiments, each node uses two threads as OpenMP threads to accelerate the process, which is the same as the configuration with WP-SGD.

6.3. *Model*

We chose hinge loss, which is used to train the support vector machine (SVM) parameters, as our objective function value. Compared with other loss functions, the contraction map rate of hinge loss is much closer to the contraction map rate of the SGD framework, i.e., $(1 - \eta\lambda)$.

It is worth noting that our work would be more conspicuous if we use a deep learning model like VGG16 [20] as an experiment benchmark. But, our paper focuses on the correctness and effectiveness of WP-SGD. There are few works on the mathematical properties of deep learning. If we use deep learning model parameters, we are not sure that the reasons for the experiment result are the intricate deep learning network unknown math properties, or the effect of WP-SGD.

6.4. *Data*

We performed experiments on KDD Cup 2010 (algebra) [26] and real-sim dataset.

In KDD Cup 2010 (algebra) datasets, the dataset's labels $y \in \{0, 1\}$ and the instances in the dataset are binary, sparse features instances. The dataset contains 8,407,752 instances for training and 510,302 instances for testing. Those instances have 20,216,830 dimensions. Most instances have about 20–40 features on average.

In the real-sim dataset, the dataset's labels $y \in \{0, 1\}$ and the instances in the dataset are binary, sparse features instances. The dataset contains 72,309 instances for training, and we randomly selected 1000 samples as test datasets. Those instances have 20,958 dimensions.

6.5. *KDD Cup 2010 (algebra) dataset experiments*

6.5.1. *Configurations:*

In all the experiment, we set $\lambda = 0.01$, $\eta = 0.0001$. In WP-SGD and D&M based WP-SGD, we use $r = 0.99999$. Because the final output is close to the zero vector, and we wanted to have more iteration steps, the initial values of all model parameters were set to 4. Then, we studied SVM model parameters and calculated the objective function value on the testing data.

In D&M based WP-SGD, different nodes allreduce their model when the fastest node exerts 1000 iterations.

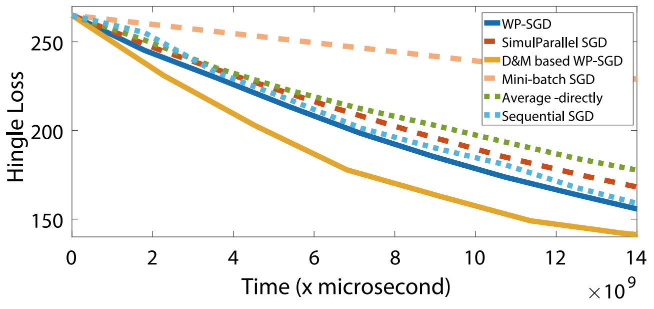

Fig. 6. The Time Performance of Different Algorithm on KDD 2010 Dataset.

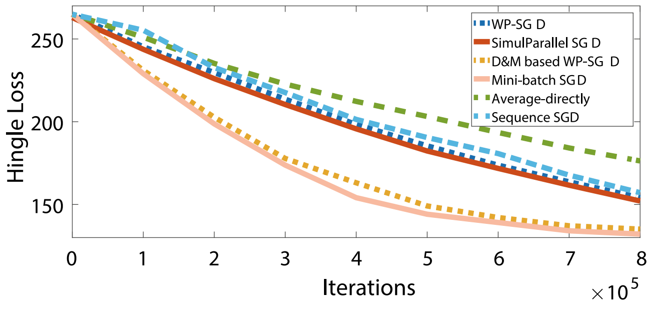

Fig. 7. The Epoch Performance of Different Algorithm on KDD 2010 Dataset.

6.5.2. *Approach:*

For SimulParallel SGD, WP-SGD and D&M based WP-SGD, in order to evaluate the convergence speed and hinge loss of the algorithms on an unbalanced-workload system, we used the following procedure: for the configuration, we trained 20 model parameters, each on an independent random permutation of a part of the whole dataset. During training, the model parameters were stored on disk after $k = 100,000 \times i$ updates of the fastest nodes.

6.5.3. *Results:*

Figs. 6 and 7 show the objective function value of different algorithms with X-axis is the number of iterations and x-axis is time.

From the aspect of the epoch, Fig. 7, mini-batch SGD is the fastest algorithm. The output of D&M based WP-SGD is close to mini-batch SGD. SimulParallel SGD, WP-SGD, and sequential SGD share almost the same convergence speed. In detail, in SimulParallel SGD, WP-SGD and sequential SGD, SimulParallel is the fastest algorithm, the output of WP-SGD is close to SimulParallel SGD. As the benchmark, the average model directly algorithm is the worst algorithm.

From the aspect of wall clock time, Fig. 6, D&M based WP-SGD is the best algorithm. Because of the burden of slow nodes, the performances of all synchronous algorithms receive the impacts: the worst algorithm is mini-batch SGD because the cost of communication is expensive. SimulParallel SGD is the worse than WP-SGD.

The above experimental results show that 1. When the system's workload is balanced, and computing resource is unlimited, the performances of the number of iterations experiment, the synchronous algorithm may be better than the asynchronous algorithm. When the system's workload is unbalanced, the performances of time-experiment, asynchronous algorithms outperform synchronous algorithms. 2. For inappropriate algorithms or synchronous algorithms on an unbalanced system, parallel technology brings negative effects.

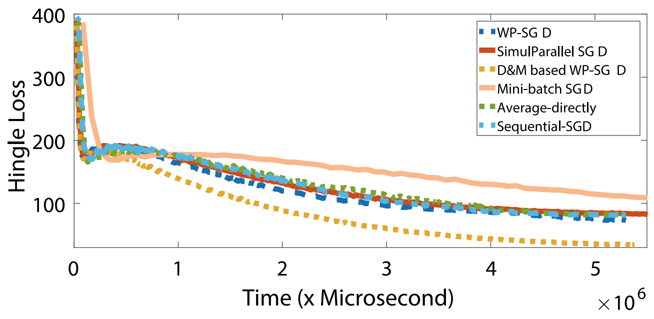

Fig. 8. The Time Performance of Different Algorithm on Real-sim Dataset.

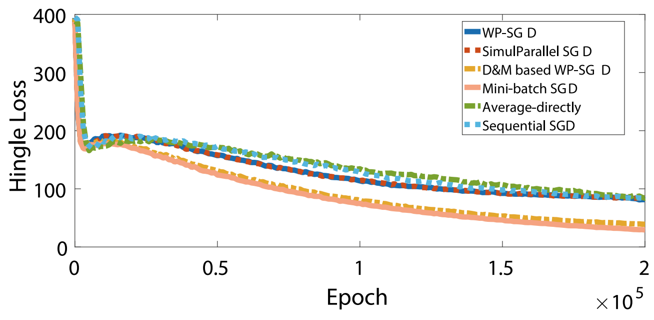

Fig. 9. The Epoch Performance of Different Algorithm on Real-sim Dataset.

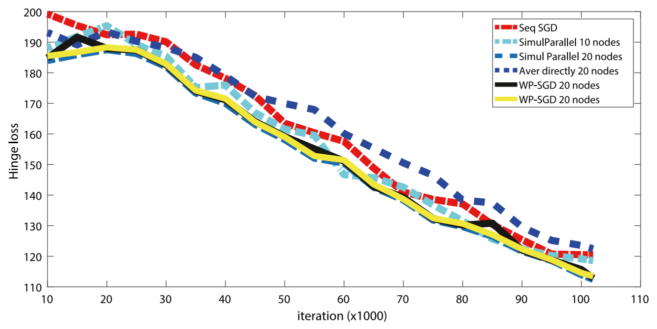

Fig. 10. Using SVM model parameters in different SGD algorithms on a cluster. In this figure, to show results clearly, we only show the key part of whole convergence process.

6.6. Real-sim dataset experiments

6.6.1. Configurations:

In the experiment, we set $\lambda = 0.001$, $\eta = 0.1$. Because the final output is close to the zero vector, and we wanted to have more iteration steps, the initial values of all model parameters were set to 100. Other settings are the same as the KDD 2010 experiments.

6.6.2. Approach:

In order to evaluate the convergence speed and hinge loss of the algorithms on an unbalanced-workload system, we used the following procedure: for the configuration, we trained 20 model parameters, each on an independent random permutation of a part of the whole dataset. During training, the model parameters were stored on disk after $k = 1000 \times i$ updates of the fastest node.

6.6.3. Results:

Figs. 8 and 9 show the objective function value of different algorithms with X-axis is the number of iterations and x-axis is time.

From the aspect of the epoch, Fig. 9, mini-batch SGD is the fastest algorithm. The output of D&M based WP-SGD is close to mini-batch SGD. SimulParallel SGD, WP-SGD, and sequential SGD share almost the same convergence speed. In detail, in SimulParallel SGD, WP-SGD and sequential SGD, SimulParallel is the fastest algorithm, the output of WP-SGD is close to SimulParallel SGD. As the benchmark, the average model directly is the worst algorithm.

From the aspect of wall clock time, Fig. 8, D&M based WP-SGD is the best algorithm. Because of the burden of slow nodes, the performances of all synchronous algorithms receive the impacts: the worst algorithm is mini-batch SGD because the cost of communication is expensive. SimulParallel SGD is the worse than WP-SGD.

The above experimental results show that 1. When the system's workload is balanced, and computing resource is unlimited, the performances of the number of iterations experiment, the synchronous algorithm may be better than the asynchronous algorithm. When the system's workload is unbalanced, the performances of time-experiment, asynchronous algorithms outperform synchronous algorithms. 2. For inappropriate algorithms or synchronous algorithms on an unbalanced system, parallel technology brings negative effects.

6.6.4. Extra experiments

For the detail of Fig. 9 is blurred between sequential SGD, SimuParallel SGD, WP-SGD, and the average model directly algorithm, we also conduct other experiments and zoom in the figure. In this experiment, algorithm settings are the same as real-sim experiments and we use doubled computing resources. We also conduct experiments which only use ten nodes as the benchmark. We think these benchmarks will show more essential characters. Our experimental results are shown in Fig. 10.

Fig. 10 shows the objective function value of sequential SGD, SimuParallel SGD, WP-SGD, and averaging the model parameters directly. To make the presentation clear, we only present part of the whole data in Fig. 10. In terms of wall clock time, the model parameters which are obtained from SimuParallel SGD clearly outperformed the ones obtained by other algorithms. The output of WP-SGD was close to the output on a balanced-workload system. Unsurprisingly, averaging the model parameters directly turned out to be the worst algorithm. The above results are consistent with Theorem 4. The convergence speeds of WP-SGD and SimuParallel SGD are the closest. Thus, on an unbalanced-workload system, WP-SGD would obtain a better objective function value in a short time.

7. Conclusion

In this paper, we have proposed WP-SGD, a data-parallel stochastic gradient descent algorithm. WP-SGD inherits the advantages of SimuParallel SGD: little I/O overhead, ideal for MapReduce implementation, superb data locality, and fault tolerance properties. This algorithm also presents strengths in an unbalanced-workload computing environment such as a heterogeneous cluster. We showed in our formula derivation that the upper bound of the objective function value in WP-SGD on an unbalanced-workload system is close to the upper bound of the objective function value in SimuParallel SGD on a balanced-workload system. Our experiments on real-world data showed that the output of WP-SGD was reasonably close to the output on a balanced-workload system.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the National Key R&D Program of China under Grant No. 2016YFB0200803, No. 2017YFB0202302, No. 2017YFB0202001, No. 2017YFB0202502, No. 2017YFB0202105, No. 2018YFB0704002, No. 2018YFC0809306; the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC01000000; the National Natural Science Foundation of China under Grant No. 61972376, No. 61502450, No. 61432018, No. 61521092; the Science Foundation of Beijing No. L182053; SKL of Computer Architecture Foundation No. CARCH3504. We also thank Dr. Fei Teng from ICT, CAS for his valuable suggestions.

Appendix

Lemma 2. *Suppose that $X^1 \dots X^k, B$ are independent distributed random variables over $\mathbb{R}^d$. Then if $A = \sum_{i=1}^k q_i \cdot X^i$ and $1 = \sum_{i=1}^k q_i$, it is the case that*

$$d(\mu_A, \mu_B) \le \sum_{i=1}^k q_i \cdot d(\mu_{X^i}, \mu_B)$$

Proof. By linearity of expectation, if X^i are independent distributed random variables then

$$\mu_A = \sum_{i=1}^k q_i \cdot \mu_{X^i}$$

Because

$$d(a, b) = \|a - b\|$$

and

$$1 = \sum_{i=1}^k q_i$$

it holds following results.

$$\mu_A - \mu_B = \sum_{i=1}^k q_i(\mu_{X^i} - \mu_B) \tag{28}$$

Considering triangle inequality, we have final result.

$$d\left(\mu_A - \mu_B\right) \le \sum_{i=1}^k q_i \left\|\mu_B - \mu_{X^i}\right\| = \sum_{i=1}^k q_i \cdot d(\mu_{X^i}, \mu_B) \quad \square$$

Corollary 1. *The fastest node consumes t data samples, $D_\eta^{t-T_i}$ is the distribution of model parameters updated after $t - T_i$ iterations in node i, and $D_\eta^{\#,t}$ is the distribution of the stochastic gradient descent update in WP-SGD.*

$$d(\mu_{D_\eta^{\#,t}}, \mu_{D_\eta^*}) \le \frac{Gk(1-\eta\lambda)^t}{\lambda \sum_{i=1}^k (1-\eta\lambda)^{T_i}}$$

Proof. Suppose $M^{\#,t}$, which is a random variable, is the output of the algorithm, and $M^{i,t-T_i}$ is the output of each node. Then

$$M^{\#,t} = \sum_{i=1}^k q_i \cdot M^{i,t-T_i}$$

based on Theorem 3, we have following result

$$d(\mu_{D_\eta^{t-T_i}}, \mu_{D_\eta^*}) \le \frac{G}{\lambda}(1-\eta\lambda)^{t-T_i}$$

Thus, using Lemma 2,

$$d(\mu_{D_\eta^{\#,t}}, \mu_{D_\eta^*}) \le \frac{G}{\lambda}\sum_{i=1}^k q_i \cdot (1-\eta\lambda)^{t-T_i}$$

Combining the above with the definition of

$$q_{1-\lambda\eta,i} = \frac{(1-\lambda\eta)^{T_i}}{\sum_{j=1}^k (1-\lambda\eta)^{T_j}}$$

we have

$$d(\mu_{D_\eta^{\#,t}}, \mu_{D_\eta^*}) \le \frac{Gk(1-\eta\lambda)^t}{\lambda(\sum_{i=1}^k (1-\eta\lambda)^{T_i})} \quad \square$$

Lemma 3. *$M^{i,t-T_i}$ is the output of node i. Then, if*

$$M^{\#,t} = \sum_{i=1}^k q_{1-\eta\lambda,i} \cdot M^{i,t-T_i}$$

then the distribution of $M^{\#,t}$ is $D_\eta^{\#,t}$. It is the case that

$$\sigma_{D_\eta{}^{\#,t}} \le \frac{\sqrt{k}}{\sum_{i=1}^k (1-\eta\lambda)^{T_i}} \left(\frac{2G\sqrt{\eta}}{\sqrt{\lambda}} + \frac{G}{\lambda}(1-\eta\lambda)^t\right)$$

Proof.

$$\sigma^2_{M^{\#,t}} = \sum_{i=1}^k q^2_{1-\lambda\eta,i} \cdot \sigma^2_{M^{i,t-T_i}}$$

Combining this with Theorem 3, we obtain

$$\begin{aligned}
& q_{1-\eta\lambda,i} \cdot \sigma_{M^{i,t-T_i}} \\
& \le q_{1-\eta\lambda,i} \cdot \left(\frac{2\sqrt{\eta}G}{\sqrt{\lambda}} + \frac{G}{\lambda}(1-\eta\lambda)^{t-T_i}\right) \\
& = \frac{1}{\sum_{j=1}^k (1-\eta\lambda)^j}\left(\frac{2G\sqrt{\eta}}{\sqrt{\lambda}} \cdot (1-\eta\lambda)^{T_i} + \frac{G}{\lambda}(1-\eta\lambda)^t\right) \\
& \le \frac{1}{\sum_{j=1}^k (1-\eta\lambda)^j}\left(\frac{2G\sqrt{\eta}}{\sqrt{\lambda}} + \frac{G}{\lambda}(1-\eta\lambda)^t\right)
\end{aligned}$$

Thus,

$$\sigma^2_{M^{\#,t}} \le \frac{k}{(\sum_{i=1}^k (1-\eta\lambda)^{T_i})^2}\left(\frac{2G\sqrt{\eta}}{\sqrt{\lambda}} + \frac{G}{\lambda}(1-\eta\lambda)^t\right)^2 \quad \square$$

Theorem 4. *Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, the bound of WP-SGD is*

$$\begin{aligned}
& E_{w\in D}[c(w)] - \min_w c(w) \\
& \le \left(\left(\frac{Gk(1-\lambda\eta)^t}{\lambda\sum_{j=1}^k (1-\lambda\eta)^{T_i}} + \right.\right. \\
& \left.\frac{\sqrt{k}}{\sum_{j=1}^k (1-\lambda\eta)^{T_i}}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}(1-\lambda\eta)^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}} \\
& \left. + \sqrt{2\eta G^2}\right)^2
\end{aligned} \tag{29}$$

Proof. Starting from Theorem 1:

$$E_{w\in D}[c(w)] - \min_w c(w) \le$$

$$\sigma_D^p\sqrt{2\|\nabla c\|_{\text{Lip}}(c(p)-\min_v(v))} + (\|\nabla c\|_{\text{Lip}}(\sigma_D^p)^2/2)+(c(p)-\min_v c(w)) \tag{30}$$

i.e.

$$E_{w\in D}[c(w)]-\min_w c(w)\le 2\sigma_D^p\sqrt{\frac{1}{2}\|\nabla c\|_{\text{Lip}}(c(p)-\min_v(v))} + (\frac{1}{\sqrt{2}}\sqrt{\|\nabla c_{Lip}\|\sigma_D^p})^2+\sqrt{(c(p)-\min_v c(w))}^2 \tag{31}$$

i.e.

$$E_{w\in D}[c(w)]-\min_w c(w)\le \left(\sigma_D^P\sqrt{\frac{\|\nabla c\|}{2}}+\sqrt{c\ (p)-\min_v c\ (w)}\right)^2 \tag{32}$$

Treat p as the machine learning model gain by WP-SGD, and we can gain

$$E_{w\in D^*}[c(w)]-\min_w c(w)\le \left(\sigma^v_{D^*_\eta}\sqrt{\frac{\|\nabla c\|}{2}}+\sqrt{c\left(\mu_{D^*_\eta}\right)-\min_w c\ (w)}\right)^2 \tag{33}$$

Using Lemma 1, we can gain following formula:

$$E_{w\in D^*}[c(w)]-\min_w c(w)\le \left(\left(\sigma_{D^{\#,t}_\eta}+d\left(\mu_{D^{\#,t}_\eta},\mu_{D^*_\eta}\right)\right)\sqrt{\frac{\|\nabla c\|}{2}}+ \sqrt{c\left(\mu_{D^*_\eta}\right)-\min_w c\ (w)}\right)^2 \tag{34}$$

Using Lemma 3 and Corollary 1, to replace the $\sigma_{D^{\#,t}_\eta}, d\left(\mu_{D^{\#,t}_\eta},\mu_{D^*_\eta}\right)$ and $\sqrt{c\left(\mu_{D^*_\eta}\right)-\min_w c\ (w)}$. We can get Theorem 4. □

Corollary 2. *For WP-SGD, when*

$$\frac{\sum_{i=1}^k r^{T_i}-\sqrt{k}}{k-\sum_{i=1}^k r^{T_i}} > \frac{r^t W_1(D^{\#,1}_\eta,D^*_\eta)}{r^t\cdot W_2(D^{\#,1}_\eta,D^*_\eta)+\sigma_{D^*_\eta}}$$

the upper bound of the objective function value of WP-SGD is closer to the minimum than is the upper bound of the objective function value of sequential SGD on the fastest nodes.

Proof. When we deduce the proof of Simul Parallel SGD (Theorem 11, Fact 28, Lemma 30, Corollary 23, Theorem 69, Theorem 70 in Zinkevich et al.'s work [29] and Lemma 1 in this paper), we would notice that the $\frac{G}{\lambda}$ is the upper bound of $W_1(D^{\#,0}_\eta,D^*_\eta)$ and $W_2(D^{\#,0}_\eta,D^*_\eta)$. In practice, $\frac{G}{\lambda}$ does not have practical application value for $\frac{G}{\lambda}$ is too large and vague for different datasets and loss function. A more tight upper bound should be described as follows:

$$E_{w\in D}[c(w)]-\min_w c(w) \le \left(\left(\frac{k(1-\lambda\eta)^t}{\sum_{j=1}^k(1-\lambda\eta)^{T_i}}W_1\left(D^{\#,0}_\eta,D^*_\eta\right)+ \frac{\sqrt{k}}{\sum_{j=1}^k(1-\lambda\eta)^{T_i}}\left(\sigma_{D^*_\eta}+(1-\lambda\eta)^t W_2\left(D^{\#,0}_\eta,D^*_\eta\right)\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|}_1 +\sqrt{2\eta G^2}\right)^2$$

Although the distribution of w_0, i.e. $D^{\#,0}_\eta$, is unobservable: the algorithm set w_0 as a fixed value, the distribution of w_1 is determined by the dataset. We can use the distribution of w_1 can be calculated by the dataset and loss function. Thus, it is practical valuable for us use the distribution of w_1 to rewrite above formula:

$$E_{w\in D}[c(w)]-\min_w c(w) \le \left(\left(\frac{k(1-\lambda\eta)^{t-1}}{\sum_{j=1}^k(1-\lambda\eta)^{T_i}}W_1\left(D^{\#,1}_\eta,D^*_\eta\right) + \frac{\sqrt{k}}{\sum_{j=1}^k(1-\lambda\eta)^{T_i}}\left(\sigma_{D^*_\eta}+(1-\lambda\eta)^{t-1}W_2\left(D^{\#,1}_\eta,D^*_\eta\right)\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|} +\sqrt{2\eta G^2}\right)^2$$

When $k=1$ and $T_i=0$ we can gain the upper bound of sequential SGD. Thus, we can gain Corollary 2 compared to above upper bounds. We can gain the final conclusion by replacing $(1-\eta\lambda)$ with r. □

Corollary 3. *For WP-SGD, when*

$$2\sum_{i=1}^k r^{T_i} > \sqrt{k}+k$$

the upper bound of the objective function value of WP-SGD is closer to the minimum than is the upper bound of the objective function value of sequential SGD on the fastest nodes.

Proof. Notice that $E_{w\in D}[c(w)]-\min_w c(w)$ decreases as the first part of Theorem 4 decreases. The first part of Theorem 4, i.e. Eq. (35), which also can be written in Eq. (36).

$$\left(\frac{Gk(1-\lambda\eta)^t}{\lambda\sum_{j=1}^k(1-\lambda\eta)^{T_i}}+\frac{\sqrt{k}}{\sum_{j=1}^k(1-\lambda\eta)^{T_i}}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda}+\frac{G}{\lambda}(1-\lambda\eta)^t\right)\right) \tag{35}$$

$$\frac{G}{\lambda}\left(\frac{k+\sqrt{k}}{\sum_{j=1}^k(1-\eta\lambda)^{T_i}}(1-\eta\lambda)^{t}+\frac{2\sqrt{k}}{\sum_{j=1}^k(1-\eta\lambda)^{T_i}}\sqrt{\eta\lambda}\right) \tag{36}$$

In addition, the sequential algorithms are a special case in WP-SGD when $k=1$. Thus, if WP-SGD is better than the sequential algorithm, the first part of Theorem 4 must be less than

$$\frac{G(1-\eta\lambda)^t}{\lambda}+\left(\frac{2G\sqrt{\eta}}{\sqrt{\lambda}}+\frac{G}{\lambda}(1-\eta\lambda)^t\right)$$

which can be written as

$$\frac{G}{\lambda}\left(2(1-\eta\lambda)^t+2\sqrt{\eta\lambda}\right)$$

It is apparent that if following inequalities hold, we obtain the result.

$$\frac{k+\sqrt{k}}{\sum_{i=1}^k(1-\eta\lambda)^{T_i}}\le 2$$

and

$$\frac{\sqrt{k}}{\sum_{i=1}^k(1-\eta\lambda)^{T_i}}\le 1$$

which means

$$2\sum_{i=1}^k(1-\eta\lambda)^{T_i} > \sqrt{k}+k$$

We can gain the final conclusion by replacing $(1-\eta\lambda)$ with r. □

Deduction 1. Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, we average parameters every s iterations for the fastest node in SimuParallel SGD. Then, the bound of the algorithm is

$$\begin{aligned}&E_{w\in D}[c(w)] - \min_w c(w)\\ &\le \left(\left(\frac{Gr^t}{\lambda} + \frac{1}{\sqrt{k}^{t/s}}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}r^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}}\right.\\ &\left.+\sqrt{2\eta G^2}\right)^2 \end{aligned} \tag{37}$$

Proof. Every averaging operation reduces the standard deviation by $1/\sqrt{k}$, and every iteration step reduces the Euclidean distance and part of the standard deviation by $(1-\eta\lambda)$. Thus, we obtain Deduction 1. □

Deduction 2. Given a cost function c such that $\|c\|_{\text{Lip}}$ and $\|\nabla c\|_{\text{Lip}}$ are bounded, we average parameters every s iterations for the fastest node in WP-SGD. Then, the bound of the algorithm is

$$\begin{aligned}&E_{w\in D}[c(w)] - \min_w c(w)\\ &\le \left(\left(\frac{Gr^t}{\lambda}\left(\frac{k}{\sum_{j=1}^{k}(1-\lambda\eta)^{T_i}}\right)^{t/s}\right.\right.\\ &\left.+\left(\frac{\sqrt{k}}{\sum_{j=1}^{k}(1-\lambda\eta)^{T_i}}\right)^{t/s}\left(\frac{2G\sqrt{\eta\lambda}}{\lambda} + \frac{G}{\lambda}r^t\right)\right)\sqrt{\frac{1}{2}\|\nabla c\|_{Lip}}\\ &\left.+\sqrt{2\eta G^2}\right)^2 \end{aligned} \tag{38}$$

Proof. Every averaging operation reduces the variance by $\frac{\sqrt{k}}{\sum_{j=1}^{k}(1-\eta\lambda)^{T_j}}$. Every iteration steps reduce the Euclidean distance and part of the variance by $(1-\eta\lambda)$. We obtain the final deduction. □

Lemma 4. *Let $c^* \ge \left\|\frac{\partial L(y,\hat{y})}{\partial\hat{y}}\right\|$ be a Lipschitz bound on the loss gradient. Then if $\eta\lambda + \eta\beta_{max}^2 c^* \le (1-\eta\lambda)^{\tau max}$ and there exist samples to make the algorithm continue at each iteration, the Algorithm 3 is a convergence to the fixed point in ℓ_2 with Lipschitz constant $1-\lambda\eta$. β_{max}^2 is defined as following formula: $\beta_{max}^2 = \max\|x^i\|^2$.*

Proof. This proof contains two parts: 1. this iterative procedure has fixed point. 2. Using any initial point, the Algorithm 3 is a convergence to the fixed point in ℓ_2 with Lipschitz constant $1-\lambda\eta$.

For the first part:

When $\tau = 0$, Algorithm 3 must have a fixed point for Algorithm 3 is degenerated into algorithm traditional sequence SGD algorithm. Traditional sequential SGD must have a fixed point (Lemma 3 in [29]). And we name the fixed point of sequential SGD as v. For the sequence of fixed point, we have $v = v_1 = v_2 = \cdots = v_n$ and following formula is still true.

$$\begin{aligned} v_{n+1} &= v_n - \eta\lambda v_{n-\tau} - \eta x^j\frac{\partial}{\partial\hat{y}}L(y^j,\hat{y})\mid_{v_{n-\tau}x^j}\\ &= v - \eta\lambda v - \eta x^j\frac{\partial}{\partial\hat{y}}L(y^j,\hat{y})\mid_{v_n x^j}\\ &= v_n - \eta\lambda v_{n-1} - \eta x^j\frac{\partial}{\partial\hat{y}}L(y^j,\hat{y})\mid_{v_n x^j}\\ &= v = v_1 = v_2 = \cdots = v_n \end{aligned} \tag{39}$$

Thus, sequential SGD's fixed point is the Algorithm 3's fixed point.

For the second part:

Firstly, by gathering terms, we obtain

$$w_{n+1} = w_n - \eta\lambda w_{n-\tau} - \eta x^j\frac{\partial}{\partial\hat{y}}L(y^j,\hat{y})\mid_{w_{n-\tau}x^j}$$

Define $u : R \mapsto \mathbb{R}$ to be equal to $u(z) = \frac{\partial}{\partial Z}L(y^i, z)$. Because $L(y^i,\hat{y})$ is convex in $\hat{y}$, $u(z)$ is increasing, and $u(z)$ is Lipschitz continuous with constant c^*.

$$w_{n+1} = w_n - \eta\lambda w_{n-\tau} - \eta x^j u(w_{n-\tau}x^j)$$

We break down w into $w_\perp$ and $w_\parallel$, and $w_\parallel$ is parallel with simple x^j, where $w = w_\perp + w_\parallel$. Thus,

$$w_{n+1\parallel} = w_{n\parallel} - \eta\lambda w_{n-\tau\parallel} - \eta x^j u(w_{n-\tau\parallel}x^j)$$

$$w_{n+1\perp} = w_{n\perp} - \eta\lambda w_{n-\tau\perp}$$

Finally, note that $d(w,v) = \sqrt{d^2(w_\parallel, v_\parallel) + d^2(w_\perp, v_\perp)}$ For the vertical dimension, because *Check* function guarantees that $w_{i\perp}$, $i = n-\tau,\ldots,n$ are to the same direction, we can gain following formula:

$$\|w_{n+1\perp}\| = \|w_{n\perp}\| - \eta\lambda\|w_{n-\tau\perp}\|$$

Thus,

$$\begin{aligned}\frac{\|w_{n+1\perp}\|}{\|w_{n\perp}\|} &= 1-\eta\lambda\frac{\|w_{n-\tau\perp}\|}{\|w_{n\perp}\|}\\ &= 1-\eta\lambda\frac{\|w_{n-1\perp}\|}{\|w_{n\perp}\|} * \frac{\|w_{n-2\perp}\|}{\|w_{n-1\perp}\|} * \cdots * \frac{\|w_{n-\tau\perp}\|}{\|w_{n-\tau+1\perp}\|} < 1-\eta\lambda\end{aligned}$$

Above requirement is guaranteed by *Check* function in Algorithm 4.

Now, we focus on the dimension parallel to x^j. We define $\alpha(w_n) = (x^j)^T w_n$ (in Algorithm 3, it is ι_n, and it is the projection of w_n on x^j), so we can know that

$$\alpha(w_{n+1}) = \alpha(w_n) - \eta\lambda\alpha(w_{n-\tau}) - \eta u(\alpha(w_{n-\tau}))\beta^2$$

This kind of delay SGD must have a fixed point (first part of this proof), and we denote this fixed point by v:

$$d(w_\parallel, v_\parallel) = \frac{1}{\beta}|\alpha(w) - \alpha(v)|$$

$$d(w_{n+1\parallel}, v_\parallel) = \frac{1}{\beta}|(w_n - \eta(\lambda\alpha(w_{n-\tau}))) - (v - \eta(\lambda\alpha(v)))|$$

Without loss of generality, assume that $\alpha(w_i) \ge \alpha(v)$ for all $i < n+1$ is true. Since $\alpha(w_n) \ge \alpha(v)$, $u(\alpha(w_n)) \ge u(\alpha(v))$. By Lipschitz continuity,

$$u(\alpha(w_n)) - u(\alpha(v)) \le c^*(\alpha(w_n) - \alpha(v))$$

Here, we define

$$\psi_n = \alpha(w_n) - \alpha(v)$$

Because of the assumption, we know that $\psi_n \ge 0$, and at the beginning, $w_0 = 0$, which means that $length_0 = 0$. The following operation is to provide a rough idea of the range of v. What we care about is the range of v closest to w_0, which we denote by $length_{\text{min}}$. τ_{max} is the maximum delay.

A rearranging of the terms yields

$$\psi_{n+1} = |\psi_n - \eta\lambda\psi_{n-\tau} - \eta\beta^2(u(\alpha(w_{n-\tau})) - u(\alpha(v)))|$$

To be able to eliminate the absolute value brackets, we need the terms in the absolute value brackets to be positive. Because $u(\alpha(w_n)) - u(\alpha(v)) \le c^*(\alpha(w_n) - \alpha(v))$ if

$$\psi_n - \eta\lambda\psi_{n-\tau} - \eta\beta^2(u(\alpha(w_{n-\tau})) - u(\alpha(v))) > 0$$

it follows that

$$\frac{\psi_n}{\psi_{n-\tau}} > \eta\lambda + \eta\beta^2 c^*$$

To satisfy the above terms, we require that

$$\frac{\psi_n}{\psi_{n-1}} > \sqrt[\tau]{\eta\lambda + \eta\beta^2 c^*}$$

Above requirement is guaranteed by *Check* function in Algorithm 4.

Above requirement can be rewritten as

$$\iota_{\min} = \frac{\iota_{t-1} - rate * \iota_{t-2}}{1 - rate}$$

Note that on the assumption, $(\alpha(w_n)) > \alpha(v)$, and so

$$\psi_{n+1} \le \psi_n - \eta\lambda\psi_{n-\tau}$$

It is apparent that ψ_n is a non-increasing series, which means that

$$\frac{\psi_{n+1}}{\psi_n} < 1 - \eta\lambda$$

It is apparent that η should satisfy

$$\eta\lambda + \eta\beta^2 c^* < \frac{\psi_n}{\psi_{n-\tau}} < (1-\eta\lambda)^\tau$$

and from the whole dataset aspect, and τ reach the maximum, η should satisfy

$$\eta\lambda + \eta\beta_{max}^2 c^* < \frac{\psi_n}{\psi_{n-\tau}} < (1-\eta\lambda)^{\tau_{max}}$$

and this then implies that

$$\begin{aligned} d(w_{n+1\|}, v) &= \frac{1}{\beta}(\alpha(w_n+1) - \alpha(v)) \\ &\le (1-\eta\lambda)\frac{1}{\beta}(\alpha(w_n) - \alpha(v)) = (1-\eta\lambda) d(w_{n\|}, v_\|) \quad \square \end{aligned}$$

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, Tensorflow: A system for large-scale machine learning, 2016.

[2] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[3] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, in: Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December, 2007, pp. 161–168.

[4] S. Chaturapruek, J.C. Duchi, C. Re, Asynchronous stochastic convex optimization: The noise is in the noise and SGD don't care, 2015, pp. 1531–1539.

[5] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems, Statistics (2015).

[6] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, Large scale distributed deep networks, in: International Conference on Neural Information Processing Systems, 2012, pp. 1223–1231.

[7] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, in: Conference on Symposium on Opearting Systems Design and Implementation, 2004, pp. 107–113.

[8] O. Dekel, R. Gilad-Bachrach, O. Shamir, L. Xiao, Optimal distributed online prediction using mini-batches, J. Mach. Learn. Res. 13 (1) (2012) 165–202.

[9] J.C. Duchi, A. Agarwal, M.J. Wainwright, Distributed dual averaging in networks, in: Advances in Neural Information Processing Systems 23: Conference on Neural Information Processing Systems 2010. Proceedings of a Meeting Held 6-9 December 2010, Vancouver, British Columbia, Canada, 2010, pp. 550–558.

[10] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (7) (2010) 257–269.

[11] N. Feng, B. Recht, C. Re, S.J. Wright, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, Adv. Neural Inf. Process. Syst. 24 (2011) 693–701.

[12] Q. Ho, J. Cipar, H. Cui, J.K. Kim, S. Lee, P.B. Gibbons, G.A. Gibson, G.R. Ganger, E.P. Xing, More effective distributed ML via a stale synchronous parallel parameter server, Adv. Neural Inf. Process. Syst. 2013 (2013) (2013) 1223–1231.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional Architecture for Fast Feature Embedding, in: ACM International Conference on Multimedia, 2014, pp. 675–678.

[14] J. Langford, A.J. Smola, M. Zinkevich, Slow learners are fast, in: Advances in Neural Information Processing Systems 22: Conference on Neural Information Processing Systems 2009. Proceedings of a Meeting Held 7-10 December 2009, Vancouver, British Columbia, Canada, 2009, pp. 2331–2339.

[15] X. Lian, C. Zhang, H. Zhang, C. Hsieh, W. Zhang, J. Liu, Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent, Neural Inf. Process. Syst. (2017) 5330–5340.

[16] U.V. Luxburg, O. Bousquet, Distance-Based Classification with Lipschitz Functions, Springer Berlin Heidelberg, 2003, pp. 314–328.

[17] A. Nemirovski, A. Juditsky, G. Lan, A. Shapiro, Robust stochastic approximation approach to stochastic programming, in: SIAM J. Optim., 2009, pp. 1574–1609.

[18] Y. Nesterov, Primal-dual subgradient methods for convex problems, Math. Program. 120 (1) (2009) 221–259.

[19] S. Shalev-Shwartz, N. Srebro, SVM optimization: Inverse dependence on training set size, in: International Conference on Machine Learning, 2008, pp. 928–935.

[20] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, Comput. Sci. (2014).

[21] H. Tang, C. Zhang, S. Gan, T. Zhang, J. Liu, Decentralization meets quantization, 2018, arXiv: Learning.

[22] T. White, Hadoop: The Definitive Guide, Yahoo! Press, 2010, pp. 1–4.

[23] J. Wu, W. Huang, J. Huang, T. Zhang, Error compensated quantized SGD and its applications to large-scale distributed optimization, Int. Conf. Mach. Learn. (2018) 5321–5329.

[24] E.P. Xing, Q. Ho, W. Dai, J.K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, Y. Yu, Petuum: A new platform for distributed machine learning on big data, IEEE Trans. Big Data 1 (2) (2013) 49–67.

[25] Y. You, Z. Zhang, C. Hsieh, J. Demmel, K. Keutzer, Imagenet training in minutes, Int. Conf. Parallel Process. (2018) 1.

[26] H. Yu, H. Lo, H. Hsieh, Feature engineering and classifier ensemble for KDD cup 2010, in: Jmlr Workshop and Conference, 2010.

[27] J. Zhang, C. De Sa, I. Mitliagkas, C. Ré, Parallel SGD: When does averaging help? 2016, arXiv preprint arXiv:1606.07365.

[28] Y. Zhang, J.C. Duchi, M.J. Wainwright, Communication-efficient algorithms for statistical optimization, 14(1) (2012) 6792–6792.

[29] M. Zinkevich, M. Weimer, A.J. Smola, L. Li, Parallelized stochastic gradient descent, Adv. Neural Inf. Process. Syst. 23 (23) (2010) 2595–2603.

Cheng Daning received his Bachelor in computer science and technology from Sun Yat-sen University, China and Ph.D. in computer architecture from Institute of Computing Technology, Chinese Academy of Sciences, China in 2014 and 2020, respectively. His research interests focus on machine learning, parallel stochastic optimization algorithm, parallel and distributed computing, and parallel and distributed deep learning.

Shigang Li received his Bachelor in computer science and technology and Ph.D. in computer architecture from the University of Science and Technology Beijing, China, in 2009 and 2014, respectively. He was a joint Ph.D. student in University of Illinois at Urbana–Champaign from Sept. 2011 to Sept. 2013 funded by CSC. He was an Assistant Professor (from June 2014 to Aug. 2018) in State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences. From Aug. 2018 to now, he is a postdoc researcher in Department of Computer Science, ETH Zurich. He is a member of ACM and IEEE. His research interests include parallel and distributed computing, parallel and distributed deep learning, and machine learning systems.

Yunquan Zhang received his B.S. degree in computer science and engineering from the Beijing Institute of Technology in 1995. He received a Ph.D. degree in Computer Software and Theory from the Chinese Academy of Sciences in 2000. He is a full professor and Ph.D. Advisor of State Key Lab of Computer Architecture, ICT, CAS. He is also appointed as the Director of National Supercomputing Center at Jinan and the General Secretary of China's High Performance Computing Expert Committee. He organizes and distributes China's TOP100 List of High Performance Computers, which traces and reports the development of the HPC system technology and usage in China. His research interests include the areas of high performance parallel computing, focusing on parallel programming models, high-performance numerical algorithms, and performance modeling and evaluation for parallel programs.