Why Dataset Properties Bound the Scalability of Parallel Machine Learning Training Algorithms

Daning Cheng, Shigang Li, *Member IEEE,* Hanping Zhang, Fen Xia, Yunquan Zhang

Abstract—As the training dataset size and the model size of machine learning increase rapidly, more computing resources are consumed to speedup the training process. However, the scalability and performance reproducibility of parallel machine learning training, which mainly uses stochastic optimization algorithms, are limited. In this paper, we demonstrate that the sample difference in the dataset plays a prominent role in the scalability of parallel machine learning algorithms. We propose to use statistical properties of dataset to measure sample differences. These properties include the variance of sample features, sample sparsity, sample diversity, and similarity in sampling sequences. We choose four types of parallel training algorithms as our research objects: (1) the asynchronous parallel SGD algorithm (Hogwild! algorithm), (2) the parallel model average SGD algorithm (minibatch SGD algorithm), (3) the decentralization optimization algorithm, and (4) the dual coordinate optimization (DADM algorithm). Our results show that the statistical properties of training datasets determine the scalability upper bound of these parallel training algorithms.

Index Terms—parallel training algorithms, training dataset, scalability, stochastic optimization methods.

1 Introduction

In the area of parallel computing theory, Gustafson's law is used to estimate the parallel speedup in the case that the workload of the parallelizable part of an application increases linearly with the compute resources. In machine learning, stochastic optimization methods, such as stochastic gradient descent and stochastic dual coordinate ascent, are usually used to train the model. Commonly, training algorithms learn models from a considerably large training dataset. To speedup the training process, most schemes use data parallelism where sample evaluation is partitioned across workers. By keeping the sample size of each worker fixed, the total workload increases proportionally with the number of workers, and thus the training throughput can be improved significantly. By now, Gustafson's law seems an ideal theoretical tool to estimate the speedup and the scalability of machine learning training based on data parallelism. However, to secure statistical reliability, the total sample size (namely, the batch size, which is related to the parallel degree) cannot always be increased since large batch sizes reduce the number of samples. Empirical results [1], [2] have shown the reduced model accuracy because of large batch size. Therefore, to guarantee the model accuracy, the scalability of parallel machine learning training algorithms is usually bounded.

In practice, the optimization algorithms are designed for general purposes. However, we notice the phenomenon that different training tasks suit different optimization algorithms. For advertisement recommendation, the asynchronous parallel methods are the mainstream. For NLP model training tasks, AllReduce manner synchronous parallel methods are more effective. What is more, current parallel and distributed optimization methods fail to work on a super-large scale computing environments with general purposes AI training tasks. Their performance does not improve too much with the increase of the computing resources [3]–[6], [6]–[14].

The reason for above theory and practice limitations is that in parallel machine learning training, in order to keep the model accuracy constant, the total workload among different workers/threads is usually a nonlinear function with the number of workers/threads. Therefore, the size of dataset, i.e., the workload, alone cannot reflect the degree of parallelism for machine learning training problems in practice. Our main target is to define and find more persuasive properties about the scalability of stochastic optimization algorithms, based on which we give the explanation to the phenomenas mentioned above.

From the perspective of optimization algorithms, the problem settings of different optimization problems are of the same type. And all of these parallel algorithms are rooted in the same sequential algorithms, i.e. SGD or SDCA. It is natural to attribute the main difference for different tasks into the difference of dataset properties, i.e., the mathematical features of objective function. Thus, in this paper, we analyze the relationship between the parallel efficiency of parallel training algorithms and the dataset statistical properties.

To achieve high efficiency, we usually hope that the benefit of parallelization is remarkable, namely, the positive effect is large with more workers being added into the training system. The above expectation is related to the degree of parallelism of a training system: A large positive effect encourages users further put more computing resources into the system. Based on the above principle and the change of objective function's value, we design a set of measurement methods to describe and measure the parallel efficiency and scalability of parallel stochastic optimization algorithms.

To make our methodology widely applied, we analyze and summarize the general mathematical properties for four different kinds of state-of-the-art parallel optimization methods: (1) the asynchronous parallel SGD algorithm, i.e., ASGD (Hogwild!

- *Daning Cheng and Yunquan Zhang were with the SKL of Computer Architecture, Institute of Computing Technology, CAS, China. E-mail: {chengdaning, zyq}@ict.ac.cn*
- *Shigang Li is with Department of Computer Science, ETH Zurich. E-mail: shigangli.cs@gmail.com*
- *Hanping Zhang and Fen Xia were with Wisdom Uranium technology Co.Ltd, Beijing. E-mail: {Xiafen, Zhang Hanping}@ebrain.ai*

(Daning Cheng and Shigang Li contributed equally to this work.)
(Yunquan Zhang is corresponding author)

algorithm) [15], (2) the parallel model average SGD algorithm (minibatch SGD algorithm[1], (3) the decentralization optimization algorithm (ECD-PSGD) [16], and (4) the dual coordinate optimization (distributed alternating dual maximization algorithm, abbr. DADM) [17].

We choose to analyze these four algorithms because they are either widely used or at the forefront of machine learning training research. Hogwild! is a centralized asynchronous training algorithm, which is implemented in mainstream training systems, such as TensorFlow [18] and MXNet [19]. Mini-batch SGD (Large-batch SGD) is the most widely used method in parallel training on HPC machines [9] [20]. ECD-PSGD and DADM stand for the main trend of optimization algorithms in current research: dual algorithm, decentralization, and data compression (quantization). We choose these algorithms also because they are well proven. We do not choose the pipeline parallel method because recent work [6] proves that it may only benefit nonsmooth problems.

Through the theoretical analysis and the experimental results, we find that when the machine learning model is fixed, the sample difference plays a vital role in the scalability. Some statistical properties of the dataset can describe sample differences. These statistical properties include (1) the variance in the sample feature in the dataset, (2) the sparsity of the sample in the dataset, (3) the diversity of the sample in the dataset, (4) the similarity of successive sampling samples. In detail, we present the following mathematical properties for scalability in a theoretical manner:

Upper Bound Conclusion: For stochastic optimization algorithms, there exists a supremum for the parallelism degree, which is decided by the statistical properties of datasets.

Sampling Conclusion: When the stochastic algorithm is fixed, the upper bound of scalability is positively related with the degree of difference between adjacent samples in the sampling sequence.

Applicability Conclusion: We propose that different algorithms suit different datasets. For DADM, Hogwild!, minibatch SGD and ECD-PSGD we have following Applicability Conclusion: DADM suits for the high $diversity$ datasets; for ASGD (such as Hogwild!), minibatch SGD and ECD-PSGD, we present them in Figure 1.

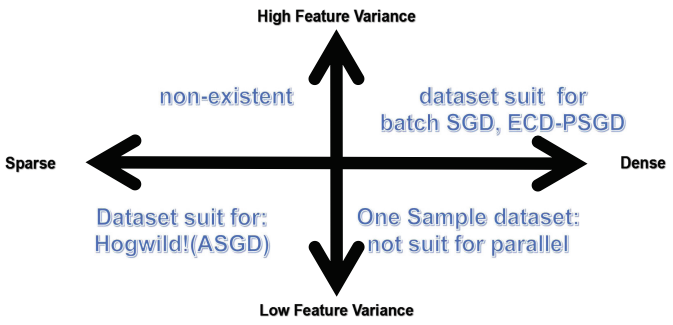


Figure 1. Different parallel training methods are applicable to different datasets.

The above conclusions about scalability show that using dataset statistical properties would give the researchers prior information to determine which parallel optimization algorithm would make full use of parallel computing resources, and break the illusion that BigData&AI applications with unlimited data would always lead to a large scale computing environment.

Our main contributions are:

- We point out that the traditional parallel theories and metrics are limited in estimating the scalability of parallel machine learning training. Dataset statistical properties, including variance of sample features, sample sparsity, sample diversity, and similarity in sampling sequences, should be used to analyze the scalability of the machine learning training tasks.
- We design a set of measurement methods to describe the parallel effect on parallel stochastic optimization methods, and analyze the upper bound of scalability for four kinds of training algorithms.
- We prove that there exsit a scalability upper bound which is decided by the datasets' statistical properties.. We also prove that shuffling dataset would help the system reach a better parallel effect.

2 Related Works

2.1 Current Scalability Problem

The current machine learning studies indeed have impressive experimental results. Although there have been a lot of successful parallel applications for machine learning training tasks [19] [18], very few research efforts are exerted on the theoretical analysis of the parallel scalability for machine learning tasks from the training dataset perspective. Especially, why a parallel algorithm works well for a specific machine learning task and whether a successful application can be transferred to other tasks are not clear. Most of the current large-scale machine learning works can be classified into the following four types: 1. Some works focus on training a specific machine learning model on a particular dataset. For example, some researchers use a specific DNN training ImageNet dataset, but their work cannot be applied to other machine learning models and datasets [9] [12] [11] [14] [13]. These works often use thousands of GPUs to train machine learning models. However, they do not show that their scalability performance can be applied to other DNNs and datasets [11], and all of their training methods cannot be proven [9] [12] [13]. For example, Thorsten Kurth et al [11] trained a Tiramisu and DeepLabv3+ CNN on a climate atmosphere dataset. For some works, the theoretical convergence result is not a tight upper bound [7], and their scalability performance needs further research. 2. Some works use pipeline parallel methods. The pipeline parallel method breaks objective functions, such as DNN, into several pieces, and these pieces are computed in a pipeline system [11]. Recent work by Igor Colin et al. [6] shows that mathematically, only nonsmooth problems may benefit from pipeline parallelization. However, in a machine learning model training process, such as a DNN training process, the nonsmooth objective function is still rare. To gain the gradient easily, most of the nonsmooth parts of the objective functions are replaced by smooth functions. For example, the step function is replaced by the sigmoid function in DNN. 3. Large-scale computing is limited in traditional math kernels in large-scale computing devices, such as GPUs. Some works attempt to optimize math kernels, such as matrix multiplication (GEMM kernel). For example, in the work by S. Chetlur et al. [10] designed methods that GPUs use to compute convolution operations. 4. For general cases, adding more parallel computing resources to these machine learning

1. In minibatch SGD, each worker computes gradients on a portion of the minibatch, and then the gradients are accumulated among the workers using all-reduce. The maximum degree of parallelism is equal to the batch size.

frameworks makes it evident that the effect of these frameworks does not improve too much. In the research by R. Anil et al [8], they claim that "it can be very difficult to scale effectively much beyond a hundred GPU workers in realistic setups." For minibatch SGD, few works attempt to use settings with a batch size larger than 32 K [9] on general-purpose machine learning training system. What is more, many research works only use less than 100 nodes in their experiment part [3]–[7].

Above facts show that there are a few works which are successfully training general-purpose AI model in super-large scale computing environments. However, with the increase of the size of dataset, i.e. the increase of the training workload, it is a matter of the utmost urgency to find why we cannot put more computing resources and gain a better performance. Further, we want to know which factors determine the maximum computing resources a training process can use, i.e. the scalability.

2.2 Parallel SGD algorithms

SGD can be dated back to the early work of Robbins and Monro [21]–[25]. In recent years, combined with the GPU and clusters [26], [27], parallelized SGD has become the most powerful weapon-solving machine learning problem [5], [7], [28]–[32]. Asynchronous parallel SGD, such as Hogwild! [33] and model average parallel SGD, such as minibatch SGD and simul parallel SGD [34], are two popular parallel SGD variants. The goal results for the asynchronous parallel SGD algorithm and sequential SGD are the same in fixed iterations. Model average parallel SGD algorithms provide the answer for how to calculate a better output in a fixed number of iterations. Decentralized parallel stochastic gradient descent [35] requires each node to exchange its own stochastic gradient and update the parameters using the information it receives [16].

2.3 Dual Coordinate Ascent Optimization

The stochastic dual coordinate ascent method (SDCA) [36] [37] is one of the most important optimization methods. Its data parallelism algorithms are a popular topic in the optimization algorithm area [38] [39]. DADM [17], DisDCA [40], and CoCoA+ [41] are the state-of-the-art parallel dual coordinate optimization algorithms.

2.4 The analysis of theoretical speedup

Many works try to give an analysis of the mathematical performance of different algorithms. In these works, the authors give a sharp analysis of stochastic optimization's parallel performance: Duchi et al. [42] show the parallel performance of sparse data on asynchronous parallel and they also give the sharp analysis about mini-batch SGD [43]. Kwangjun Ahn et al. give a sharp analysis of the relationship between SGD convergence speed and shuffling data [44]. Other researchers also focus on different stochastic optimization algorithms' mathematical properties, like stochastic incremental methods.

However, these works mainly focus on a single mathematical property on one algorithm [42], [43], [45]. They do not focus on the parallel performance and list the different algorithms' performance analysis from the view of the same mathematical properties.

3 Problem Setup and Notations

3.1 Problem Setting

An optimization method is used to solve the following minimum problem:

$$min\hat{f}(x) = \mathbb{E}_{\Xi}F(x;\Xi)$$

where Ξ is a random variable that satisfies a certain distribution. In most cases, the distribution of Ξ is unknown or cannot be presented as a formula form. It is common that we use a frequency histogram to replace Ξ's PDF. The above formula is written as:

$$min\hat{f}(x) = \mathbb{E}_{\Xi}F(x;\Xi) \approx f(x) = \frac{1}{n}\sum_{i=1}^{n}F(x;\xi_i) \quad (1)$$

where ξ_i is the sample sampled from Ξ. The collection of $\{\xi_1, \xi_2, ..., \xi_n\}$ is the dataset. In addition, $x^* = argmin\, f(x)$.

For regularized risk minimization, in many cases, $F(x;\xi_i)$ is presented as the following formula [34]:

$$F(x;\xi_i) = L(\xi_i, x) + \lambda\psi(x)$$

$L(\xi_i, x)$ is the loss function, such as hinge loss for the SVM model and logloss for the LR model, and $\psi(x)$ is the regulation function. Usually, $\psi(x) = \frac{1}{2}\|x\|^2$, i.e.,

$$F(x;\xi_i) = L(\xi_i, x) + \frac{\lambda}{2}\|x\|^2 \quad (2)$$

3.2 Assumptions

3.2.1 Sample Uniformly Distributed Assumption

To make the analysis simple and clear, we make the assumption that the distributions of the sample's feature value and nonzero feature's position are uniform. As we can see, most of the datasets satisfy this assumption. We have to use this assumption because, without it, common definitions may blur the boundary between different types of datasets, as shown in the following example.

Example 1. Considering the dataset: {(1,0,0...,0), (2,0,0...,0), (3,0,0...,0) ... (dataset_size,0,0...,0)}, under the common definitions, this dataset is a sparse dataset. However, after deleting unused features, the above dataset is a dense dataset.

3.2.2 The positive correlations between the mathematical properties of stochastic gradient and samples

Although the specific functions between stochastic gradient and samples are various and cannot be present in a unified form, it is easy to see that the statistical mathematical properties of stochastic gradient is positive correlated to the statistical methematical properties of samples in dataset. These statistical mathematical properties include sparsity, diversity, variance and so on.

This positive correlations is the base for us to map sample statistical properties into stochastic gradient properties in algorithm analysis.

Example 2. For linear model like SVM and LR, the sparsity of current sample is equal to the sparsity of model's stochastic gradient based on current sample. For convolution layer in DNN, a sparse input is apt to produce a sparse backpropagation gradient.

3.2.3 CRCW PRAM model

First of all, we have to build a model where we measure the speedup. We want to prove that the limitation of speedup is rooted in the parallel stochastic optimization algorithm and dataset. Therefore, to avoid the discussion of the code implementation, parallel math kernel implementation and hardware setting, we assume that the workers in a cluster have unlimited memory and bandwidth in the network, which is CRCW PRAM model in parallel research.

Iteration and time under the CRCW PRAM model. We also conduct experiments under the CRCW PRAM model because it is easy to map the number of system/parameters server iterations, $number_{iteration}$, to the real-time. For synchronous parallel algorithms such as minibatch SGD, ECD-PSGD, and DADM, the time for one iteration on a single worker (thread, CPU....) is t_{single}, and the time for a system with m workers is $t_{single}*number_{iteration}$. For asynchronous parallel algorithms such as Hogwild!, the time for one iteration on a single worker (CPU) is t_{single}, and the time for a system with m workers is $t_{single} * \frac{number_{iteration}}{m}$. Thus, in experiments, in regard to exhibiting the time consumption performance under the CRCW PRAM model, it is enough to exhibit the number of iterations.

3.3 Dataset Statistical Properties

3.3.1 The Local Similarity of Consecutive Samples in the Sampling Sequence, i.e., $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$, for Different Algorithms

We find that the similarity of consecutive samples in sampling sequences is important because in online learning applications, which often use SGD as their optimization method, rearranging samples always leads to a better speedup performance. In an online learning application, the samples in the sample sequence are often similar to their neighborhood samples. For example, the online sample from an advertisement click is similar to its neighborhood because user interest cannot be changed drastically.

To make our presentation clear, we have to define the local similarity, i.e., $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$, for algorithm $\mathcal{A}$ on dataset $\mathcal{D}$ sampling sequence $\mathcal{S}$ is used.

Before we define the local similarity of consecutive samples in sampling sequences, i.e., $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$, we have to define the value of C_sim.

For a sampling sequence $\xi_1, \xi_2 ...., \xi_n$ and a range $range$, C_{sim} is defined as

$$C\_sim_{range} = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{range-1}\left\|\xi_i - \xi_{(i+j)\%length}\right\|_0}{range} \quad (3)$$

where $length$ is the sequence length.

For a sample collection $\{\xi_1, \xi_2.., \xi_n\}$, their different sampling orders have different C_sim values.

Example 3. For datasets {(0,0,0), (0,0,1), (0,1,1), (0,1,0), (1,1,0), and (1,0,0)}, the samples have 2 different C_sim_2 sequences:
1. Sequence with C_sim_2=0.5: (0,0,0), (0,0,1), (0,1,1), (0,1,0), (1,1,0), (1,0,0)
2. Sequence with C_sim_2=1: (0,0,0), (1,1,0), (0,0,1), (1,0,0), (0,1,0),(0,1,1)

Based on C_sim_{range}, we can define $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$.

When $\mathcal{A}$ is an asynchronous SGD such as Hogwild!, t$\mathcal{S}$ is $\xi_1, \xi_2, ..., \xi_t$, $\xi_i \in \mathcal{D}$, and the lag between when a gradient is computed and when it is used is always less than or equal to τ_{max}. Then, $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S}) = C\_sim_{\tau_{max}}$ on $\mathcal{S}$.

When $\mathcal{A}$ is a synchronous algorithm, such as DADM, minibatch SGD and ECD-PSGD, $\mathcal{S}$ is $[\xi_1, \xi_2, ..., \xi_{batch\_size}]$, $[\xi_{batch\_size+1}, \xi_{batch\_size+2}, ..., \xi_{2batch\_size}]$,..., $\xi_i \in \mathcal{D}$, where the samples or gradients in $[\cdot]$ are in one batch. We use the following two steps to calculate $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$: 1, for the samples in a batch, we find the sequence that consists of these samples. This sequence's C_sim$_{batch_size}$ is larger than that for any sequences that consist of these samples. We name C_sim$_{batch_size}$ for this sequence C_sim_batch. 2, for the whole sampling sequence $\mathcal{S}$, we choose the batch whose C_sim_batch is the maximum in $\mathcal{S}$. $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is this batch's C_sim_batch.

3.3.2 Feature Variance and Sparsity

In this paper, we define the variance in feature k as

$$fm_k = \frac{1}{n}\sum_{i=1}^{n}\xi_i(k)$$
$$fv_k = \frac{1}{n}\sum_{i=1}^{n}(\xi_i(k) - fm_k)^2$$

where $\xi_i(k)$ is the k-th feature in ξ_i, fm is the feature mean and fv is the feature variance.

The sparsity is the rate between the number of zero elements and the size of the sample. The density is $1-$ sparsity.

It is clear that when the dataset is sparse, the feature variance must be small.

3.3.3 Diversity

The diversity is the number of different kinds of samples in the dataset. We notice that the size of the dataset may be large, but the dataset is the replication of several samples.

Diversity cannot be present by variance and sparsity. Thus, it is necessary to use this metric to describe the sample difference. In the following example, we show that low variance and low-density datasets can still have high diversity.

Example 4. Low-density dataset whose sample size is large and diversity is high: (1,0,0...,0), (0,1,0...,0), ..., (0,0,0...,1).

Example 5. The diversity of the low variance dataset {(0.01),(0.02),(0.03)...,(0.99),(1)} is higher than the diversity of the high variance dataset {(100),(-100),(100),(-100)...,(100),(-100)}.

4 Theoretical Analysis

Because the convergence analysis about different algorithms are different and the upper bound analysis from different analysis methods about the same algorithm is also various, for example, the analysis about asynchronous parallel SGD in different works [4], [5], [7], [15], it is impossible to gain the form about speedup's function even for a specific algorithm. Thus, we have to focus on general mathematical properties about scalability and parallel speedup.

4.1 Main Conclusion

Our direct conclusions are presented in introduction part. In this part, we will show how to gain these conclusions.

4.2 The Upper Bound of Scalability

4.2.1 The Method to Measure the Scalability of Machine Learning Algorithm

Traditional definition of scalability cannot cover the situation of machine learning training algorithm. Traditional definition of scalability mainly focuses on the whole workload and the workload on per worker. The total workload among different workers/threads is usually a linear function with the number of workers/threads. Strong scalability focuses on the change of workload per worker when the whole workload is fixed. Weak scalability focuses on the change of whole workload when the workload per worker is fixed. And on CRCW PRAM, an ideal system, discussing scaling efficiency is meaningless.

However, in machine parallel learning training algorithm, the whole workload and the workload per worker are various depending on the number of workers: In two systems with the different number of workers and the same parallel machine learning training algorithm, we cannot expect that they can achieve the same objective function value with the same amount of samples.

What is more, in practice, the targets of training process are different: 1.Under the condition that training model into a fixed accuracy, training time should be as short as possible. This case suits the situation where the dataset and model are fixed, and the model does not need to be update; 2. Under the condition that training time is limited, the value of objective function should as small as possible. This case suits the situation where the dataset and model are dynamic and the model should be update frequently, like advertisement recommend system.

Because of the unavailability of traditional scalability, we have to redefine the scalability in parallel machine learning training algorithm. Our scalability builds the relationship between the number of workers, system's whole workload, workload per worker and the value of objective function: By analogy with the definition of strong scalability, we define the "cost": The cost is the number of iterations for each worker when the system reaches convergence point. By analogy with the definition of weak scalability, we define the "gain": The gain is the value of the objective function at system's a fixed iteration. To describe the scaling efficiency in CRCW PRAM, we have to define new concepts, the gain-growth: First gain-growth definition: The gain-growth is the difference between the cost. Second gain-growth definition: the gain-growth is the value of the objective function's difference.

It is worth noting that in this definition, the second gain-growth for ASGD is always negative. Following examples show how to calculate two kinds of gain-growth in different algorithms.

Example 6. When the system uses a real-sim dataset and eight equal performance workers on other stable algorithm settings on the Hogwild! algorithm to train LR model, the server uses 6,242 iterations to reach the point of convergence. In this case, the cost is the number of iterations for each worker: 6,242/8 = 781 iterations per worker. And at 1,497th iteration, the logloss of the system is 0.2974. When the system uses the real-sim dataset and nine equal performance workers, the server uses 6,497 iterations to reach the point of convergence. In this case, the cost is the number of iterations per worker: 6,497/9 = 722 iterations per worker. And at 1,497th iteration, the logloss of the system is 0.3057. Thus, the first gain-growth is 781 - 722 = 59 iterations. And the second gain-growth is 0.2974 - 0.3057 = -0.0082. As we can see from this example, although the server has to train more iterations, the number of iterations per worker decreases.

Example 7. Using the HIGGS dataset, 2 workers and other stable algorithm settings on the minibatch SGD algorithm to train LR model, the server uses 106 iterations to reach the point of convergence. In this case, the cost is the number of iterations for each worker: 106 iterations per worker (the number of the iteration for system is equal to the number of the iteration for per worker in minibatch SGD). And at 50 server iterations, the logloss for this model is 4.7525. Using the HIGGS dataset, 3 workers and other stable algorithm settings on the minibatch SGD to train LR model, the server and each worker use 97 iterations to reach the point of convergence. And at 50 server iterations, the logloss for this model is 4.5871. The first gain-growth is 106 - 97 = 9, and the second gain-growth is $4.7525 - 4.5871 = 0.1654$.

In fact, the above gain-growth definitions are two aspects of one phenomenon. In practice, we can use them both in one algorithm. Apparently, in one algorithm, the values of gain-growth in the above situations are positively related.

However, we use the above two situations in our paper for the following two reasons: 1. The proof methods for algorithms' theory are different. Different algorithms are proven in different aspects. It is easy to prove synchronous algorithms in second situation and asynchronous algorithms for the first situation. So, we kept them all. 2. In different presentation methods for experimental data, the above two situations present different clarity: the first case is appropriate for a table, and the second case is appropriate for a chart (which can be shown easily by the gap between different convergence curves).

Therefore, based on the above reasons, in theory, the upper bounds of synchronous algorithms are defined as the second case, and ASGD is defined as the first case. In experiments, chart results are presented in the second case and table results for the first case.

4.2.2 The Theoretical Upper Bound of Algorithm Scalability

Based on the definition of gain and gain-growth, under the CRCW PRAM model, the upper bound of algorithm scalability, m_{max}, describes the following two situations:

1. Under the CRCW PRAM model, with the increase number of workers at the range $[m_{max}, \inf]$, the second gain-growth is positive but close to zero. In this case, the gain-growth does not cover the parallel cost on a real computer. This situation is appropriate for minibatch SGD, DADM, and ECD-PSGD.

Example 8. Using the real-sim dataset and other stable algorithm settings on minibatch SGD, the second gain-growths at 15,000 iterations are 0.0011, 0.0006, 0.0003, 0.0002, and 0.00018, matching the algorithm settings whose numbers of workers/batch sizes are 14, 15, 16, 17, 18, and 19. As we can see from this case, the gain-growth decreases (to zero). Thus, when the growth cannot cover the parallel cost, the system meets its speedup upper bound.

2. Under the CRCW PRAM model, with the increase number of workers at the range $[m_{max}, \inf]$, the first gain growth is negative (i.e., cost increases drastically). This situation is appropriate for algorithms such as Hogwild!.

Example 9. when the system uses the HIGGS dataset and other stable algorithm settings on the Hogwild! algorithm, the first gain-growth is 14, 4, -7, -39, and -72, matching the algorithm

settings whose number of workers is 3, 4, 5, 6, and 7. As we can see from this case, the gain-growth decreases. Thus, when the first gain-growth is negative, the system meets its speedup upper bound.

4.3 Analysis Sketches

We offer a brief analysis in this part and whole analysis in appendix.

4.3.1 Key Part in Different Algorithm's Analysis

For different algorithms, their parallel principles are different.

For Hogwild!, the key part is the stochastic gradient's feature conflict during asynchronous parallel delay: At jth iteration, workers pull model, calculate current stochastic gradient $gradient_j$ and push stochastic gradient at system's $j+\tau$th iteration. The more $gradient_j$ shares non-zero features with other stochasic gradients in τ iterations, the more iterations the system has to use.

For mini-batch SGD and ECD-PSGD, the key part is the variance of the machine learning model. The upper bound of the loss function is positively related to the model variance.

For DADM, the key part is the sub-problem difference between different workers. The more different these sub-problems are, the better the parallel effect of DADM is.

4.3.2 Applicability conclusion

Based on the key part of the proof, we show a brief theoretical analysis introduction: For Hogwild!, the stochastic gradient sparsity makes less feature conflict. The sparsity of stochastic gradient is positively related to the sparsity of the sample in the dataset; For minibatch and ECD-PSGD, the stochastic gradient variance is positively related to the sample variance in the dataset. The model is the accumulation of stochastic gradient; For DADM, just as their proof method shows, if two workers share the same sub-problem, the parallel technology is invalid.

4.3.3 Sampling conclusion

Sampling conclusion is can also be conducted from the key part of proof: For Hogwild!, if the value of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small, the samples in sampling sequence would share a lot of non-zero features with their neighborhood samples, which would lead to conflict features. For ECD-PSGD and mini-batch SGD, if the value of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small, the variance of the sample is small (because only a few features contribute variance), which leads to the reduction of the model variance and low parallel efficiency. For DADM, if the value of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small, the samples, which are training in an iteration, are almost the same. The above fact shows that the different workers are solving almost the same sub-problems, which would negatively affect parallel technology.

4.3.4 Upper bound conclusion

The basic analysis method is to check the convergence lemma's change when the parallel variant is various. In Section 4.2, we summarized two upper bound situations. In analysis, the main task is to check how the algorithms reach the upper bound and which factors influence the speed of algorithm reaching the upper bound.

4.4 Parallel Machine Learning Training Algorithm and Traditional Parallel Algorithm

In regard to the speedup of the algorithm, the current machine learning training algorithm and traditional application show great differences.

For traditional problems, we have the following claims.

1. The accuracy of the output is determined by the number of grids. Usually, in many cases, these problems can be transferred into linear algebra problems such as stencil and matrix multiplication. The larger the problem size, the more accurate the results.

Example 10. In stencil applications, such as atmosphere simulation applications, the more grids we have, the more accurate the results we can obtain.

2. The number of grids and problem size determine how many workers this application can use, which is shown in Amdahl's Law. The upper bound of theoretical speedup and real computer environment determine the real upper bound of speedup.

Example 11. When using multithread parallel methods to compute a 10*10 vector inner product on a server, which consists of 2 Intel ® Xeon® CPU E5-2680 2.88 GHz, i.e., 20 cores together, we used approximately ten cores at most (the upper bound of theoretical speedup). However, considering the parallel cost, in a real situation, we only use one core to solve this problem, i.e., the real upper bound of speedup.

However, for parallel stochastic optimization algorithms, we make the following claims.

1. The accuracy of the output is determined by the size of the dataset. Based on the asymptotics in statistics [46], in Eq. 1, if the size of training dataset is larger, the gap between $\hat{f}(x)$ and $f(x)$ would be closer.

Example 12. In the real-sim dataset, the log loss on a test dataset of the LR model, which is trained by part of the training dataset, is higher than the LR model, which is trained by the whole training dataset.

2. The statistical properties of the dataset influences the characteristics of the machine learning model (objective function in stochastic optimization problem). The mathematical properties of the objective function determine the upper bound of speedup. Usually, the size of the dataset may influence the upper bound of speedup. However, the size of the dataset is not the decisive element, as in the following example.

Example 13. **One sample dataset:** Considering the dataset that only contains one sample, the size of the dataset can be any number by duplicating this sample. However, the training machine learning model on this dataset cannot be accelerated by any parallel stochastic optimization algorithm.

5 EXPERIMENT

5.1 Experimental Setting

5.1.1 Dataset

We choose a sparse dataset with small feature variance: real-sim dataset and a dense dataset with large feature variance: HIGGS dataset. Detailed information about the above datasets is shown in Table 1. Their suitable algorithms are shown in Figure 2. In all cases, the dataset is randomly split into two parts: a training set containing 80% of the dataset samples and a valid set containing 20% of the dataset samples.

Table 1
Dataset information

dataset	#features	#size	feature variance	density
real-sim (High diversity dataset)	20,958	72,309	(0,1)	$< 3\%$
HIGGS	28	11,000,000	(-4,3)	100%
Simulated Data for Hogwild! upper bound experiments	20,958	-	0/1	70%
Simulated Data: Small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dense dataset	28/1000	-	< 9	100%
Simulated Data: Large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dense dataset	28/1000	-	< 0.25	100%
Simulated Data: Small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ sparse dataset	20,958	-	< 0.25	$< 3\%$
Simulated Data: Large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ sparse dataset	20,958	-	< 0.25	$< 3\%$
Part Real-sim dataset (Low diversity dataset)	20,958	72,309	< 0.25	$< 3\%$
Part Real-sim dataset (Middle diversity dataset)	20,958	72,309	< 0.25	$< 3\%$

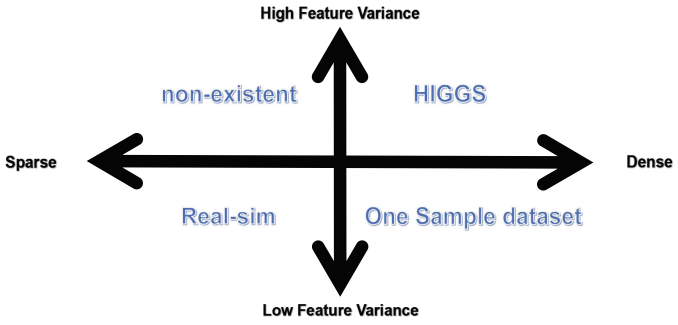


Figure 2. The best performance dataset for different algorithms

To match our theory, we also build three groups (nine in all) simulated datasets: (1) normal dataset for upper bound experiments, (2) different $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ datasets, and (3) different sample diversity datasets. The samples in these datasets are generated randomly, and the label is generated by the function $label_i = sign(\xi_i \cdot ruler)$, where $ruler$ is the vector $(-1, 2, -3, 4..., (-1)^{sample\_size} * sample\_size)$. The design of simulated datasets subjects to the definition of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ and sample diversity.

Small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset and large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset The small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset and large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset are used to match the $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ related theory. All information is shown in Table 1. All sample features in this group are sampled from the same distribution. In the experiments, we use a uniform distribution $\mathcal{U}(range\_begin, range\_end)$, where the range is shown in Table 1.

In the $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ experiments, the size of the test dataset is 25% of the number of training data (20% in the whole dataset). The data in the test data share the same feature distribution and density characteristic as the training data. In a small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dense dataset, we name it as $\mathcal{D}_1$, the sample offered by the t-th iteration is modified by the sample at the $t-1$-th iteration: we randomly choose 30% features and randomly change these features' values. And we name the sampling sequence as $\mathcal{S}_1$. In a large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dense dataset, $\mathcal{D}_2$, the sample offered by the t-th iteration is modified by the sample at the $t-1$-th iteration: we randomly choose 90% features and randomly change these features' values. And we name the sampling sequence as $\mathcal{S}_2$. In a small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ sparse dataset, $\mathcal{D}_3$, the sample offered by the t-th iteration is modified by the sample at the $t-1$-th iteration: we randomly choose 30% features and randomly change these features' values. To make the sample sparse, we also randomly pick some features and set them as zero, and the sparsity is equal to the sparsity of the sample at the first iteration. And we name the sampling sequence as $\mathcal{S}_3$. In a large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ sparse dataset, $\mathcal{D}_4$,the sample offered by the t-th iteration is modified by the sample at the $t-1$-th iteration: we randomly choose 90% features and randomly change these features' values. To make the sample sparse, we also randomly select some features and set them as zero, and the sparsity is equal to the sparsity of the sample at the first iteration. And we name the sampling sequence as $\mathcal{S}_4$. Apparently, for an algorithm $\mathcal{A}$, we have $LS_{\mathcal{A}}(\mathcal{D}_1,\mathcal{S}_1) \leq LS_{\mathcal{A}}(\mathcal{D}_2,\mathcal{S}_2)$, and $LS_{\mathcal{A}}(\mathcal{D}_3,\mathcal{S}_3) \leq LS_{\mathcal{A}}(\mathcal{D}_4,\mathcal{S}_4)$

As we can see from the above dataset design setting, when the sparsity of the two datasets is the same and the size of the dataset is large enough, the two datasets are the same.

Simulated dataset for upper bound experiments Our experimental environment is poor, and it can only support 24 workers at once.

The upper bound of Hogwild!'s speedup on real-sim dataset exceeds the number of cores of our computing environment. Therefore, we have to build a simulated dataset whose upper bound of speedup is easy to reach. In our simulated dataset, the density is 70%. The feature distribution is the same as $LS_{\mathcal{A}(\mathcal{D},\mathcal{S})}$ experiments. Other information is shown in Table 1. Because of the limitation of the maximum number of workers, the DADM experiments also have to build a dataset to match the experimental results. In this experiment, we use a randomly selected 1/8 real-sim dataset as our training dataset and a randomly picked 1/40 real-sim dataset as our test dataset. In scalability upper bound experiments, the size of the test dataset is 20% of the number of training data. The data in the test data only share the same feature range and density characteristic with the training data.

Simulated dataset for sample diversity experiments Measuring the sample diversity is a time-costing job. We only get the diversity data about real-sim dataset, which is about 72250. Thus, we use the following method to build three different sample diversity datasets. The Real-sim dataset is the high diversity dataset in the experiment.

We equally cut the real-sim dataset into 4 parts:
$\{sub\_dataset_1, sub\_dataset_2, sub\_dataset_3, sub\_dataset_4\}$
The sample diversity of the sub_dataset is lower than that of the whole dataset. Thus, we build the middle diversity dataset, abbr. real_sim$_2$, whose the diversity is about 36000, as follows:
$\{sub\_dataset_1, sub\_dataset_1, sub\_dataset_2, sub\_dataset_2\}$
We also build the low sample diversity dataset, abbr. real_sim$_4$, whose the diversity is about 18000, as follows:
$\{sub\_dataset_1, sub\_dataset_1, sub\_dataset_1, sub\_dataset_1\}$
Note: It is appropriate to using simulated dataset:(1) Some statistical values are hard to control and we have a limitation of

computing resources. In local similarity experiments, it is costly to build two sequences that have remarkably different $LS_{\mathcal{A}}$ in one dataset. In diversity experiments, calculating diversity for each dataset is also a time-costing job. (2) It is appropriate to tease out the answer. It is hard to find two datasets that are almost the same without target mathematical properties, which are used in control variant experiments. Simulated datasets can build almost the same datasets, and these datasets are only different in target mathematical properties. Control variant experiments would help experiments match theories. Thus, we had to build our own experimental datasets.

5.1.2 Machine Learning Model

We, of course, know that using a popular machine learning model, such as CNN or DNN, can make our paper extremely impressive. However, the mathematical properties of a complex machine learning model, such as CNN or DNN, are unknown and unstable. If we use CNN or DNN as our experimental machine learning model, there will be a question of whether the reasons for our experimental results are unknown mathematical properties or the properties that are offered in our paper. Thus, it is necessary to use a safe and transparent machine learning model with known mathematical properties.

In our experiment, we solve the problem of training the L2 norm logistic regression model because the log loss function is applicable to all requirements that are asked by Hogwild!, minibatch SGD, DADM, and ECD-PSGD. Moreover, the L2 norm logistic regression model is currently the mainstream machine learning model used in advertisement recommendation tasks.

The logistic loss function is shown in Eq. 4.

$$\underset{x}{argmin}\ \frac{1}{n}\sum_{i=1}^{n}\Phi(lable_i * \xi_i \cdot x) + \lambda/2\left\|x\right\|^2 \qquad (4)$$

where Φ is the logistic loss, i.e., $\Phi(t) = log(1 + e^{-t})$ and $\lambda = 0.01$.

5.1.3 Model's Measurement and Evaluation

In the feature variance and sparsity experiment, diversity experiment and $LS_{\mathcal{A}}(\mathcal{D}, \mathcal{S})$ experiment, we use gain to indicate gain-growth.

In these experiments, because of the CRCW PRAM model, we show the convergence curve on the figure whose X-axis is the number of iterations, and the Y-axis is the logloss. In our figure, the gap can indicate the effect of parallel technology. The upper bound of algorithm speedup has two situations. Therefore, different algorithms have different metrics to determine the speedup effect of the parallel algorithm.

For ASGD, i.e., Hogwild!, the effect is better when the gap is smaller, for ASGD's second gain-growth definition is always negative. When the gap is small, the number of system iterations to reach a fixed ϵ is stable when increasing the number of workers. Then, the number of iterations in each worker decreases. For ECD-PSGD, DADM and minibatch SGD, the effect is better when the gap is large because the synchronous first gain-growth definition is always positive. When the gap is large, at the fixed iteration, the log loss from a particular algorithm worker setting is smaller.

In upper bound experiments, we use cost to indicate gain-growth in table form. In our upper bound experiments, the iterations per worker can indicate the upper bound of speedup.

5.2 Feature Variance and Sparsity Experiment

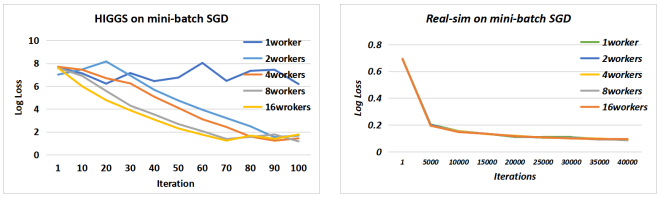
5.2.1 Algorithm Setting

In this experiment, we run HIGGS and real-sim on different algorithms to make the comparison.

In Hogwild!, the learning rate is 0.1. In the minibatch SGD and ECD-PSGD, learning rates are 0.1. In Hogwild! experiments on the HIGGS dataset, to gain a stable curve, we set the minibatch as 4. In the ECD-PSGD experiment, we connect all workers into a ring, and we do not compress the data.

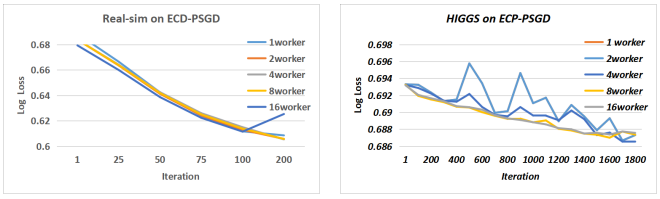
5.2.2 Experimental Results and Experimental analysis

The experimental results are shown in Figure 3, 5, 4. In our feature variance and sparsity experiment, our experimental results match well with the theoretical analysis: Our experimental results match the Figure 1. (1) In minibatch SGD and ECD-PSGD, the parallel effect is remarkable for the large variance dataset (HIGGS). Large batch setting minibatch SGD converges faster. However, for the sparse dataset (real-sim), the parallel technology does not exert any influence on the convergence speed. For ECD-PSGD, parallel technology has a negative impact. (2) For the ASGD algorithm, i.e., Hogwild!, with the increase number of workers, the influence on convergence speed is minor on the sparse dataset. The iteration number on each workers decrease linearly. However, for the feature variance dataset (HIGGS), the convergence speed drastically decreases, which means that the iteration number on each worker is not obviously reduced. In some cases, the iteration number on each worker increases as the number of workers increases.



(a) HIGGS on minibatch SGD (b) Real-sim on minibatch SGD

Figure 3. The performance of different datasets on minibatch SGD.



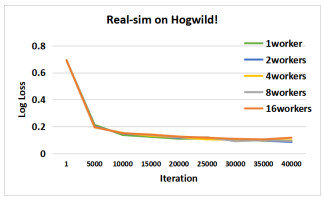
(a) Real-sim on ECP-PSGD (b) HIGGS on ECD-PSGD

Figure 4. The performance of different datasets on ECD-PSGD.

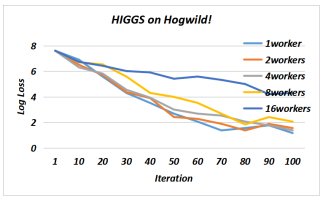
5.3 Sample Diversity Experiment

5.3.1 Algorithm Setting

To present our experiments clearly, we use 1 worker to 16 workers to train the real_sim, real_sim$_2$ and real_sim$_4$ datasets. In each worker, the minibatch size is one.

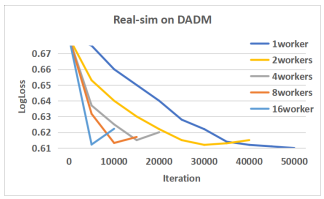


(a) Real-sim on Hogwild!

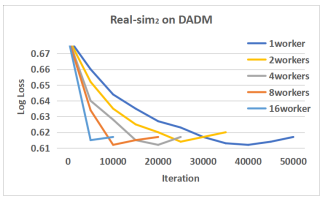


(b) HIGGS on Hogwild!

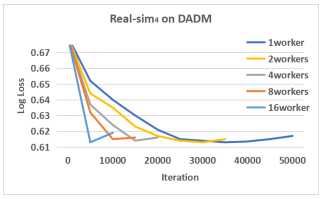
Figure 5. The performance of different datasets on Hogwild!. In this case, the effect is better when the gap is small. The number of iterations for the server to reach a fixed ϵ is stable when increase the number of workers. Then, the number of iterations in each worker will decrease.



(a) Real-sim dataset on DADM



(b) Real-sim$_2$ dataset on DADM



(c) Real-sim$_4$ dataset on DADM

Figure 6. The performance of different sample diversity datasets on DADM. The gap order is Real-sim $>$ Real-sim$_2$ $>$ Real-sim$_4$, which means parallel effect (second gain-growth) order is Real-sim $>$ Real-sim$_2$ $>$ Real-sim$_4$

5.3.2 Experimental Results and Experimental analysis

The experimental results are shown in Figure 6. In our sample diversity experiment, our experimental results match well with the theoretical analysis: high sample diversity leads to better scalability. In DADM, when sample diversity is large, at the same iteration, we can obtain more gain-growth: the gap between the different lines is large.

In this experiment part, the absolutely convergence speeds are not comparability: The training dataset they use are different, which means their objective functions are different. The same algorithm would show the different performance on different problems.

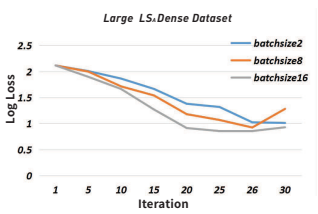
What we want to compare is the gap between different convergence lines. The gap indicates that the benefit which we gain from the parallel. The larger the gap between different curves is, the higher the parallel efficiency is.

5.4 $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ Experiment

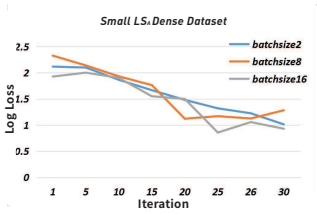
5.4.1 Algorithm Setting

The algorithm setting in this section is the same as the feature variance and sparsity section. The above sections show that different datasets suit different algorithms; we only present 1. Sparse dataset for Hogwild! 2. Feature variance dataset for minibatch SGD (#feature is 28) and ECD-PSGD (#feature is 1000). In Hogwild! and the DADM experiment, the first sample is sampled from the real-sim dataset. In the minibatch SGD experiment, the first sample is sampled from the HIGGS dataset. In the ECD-PSGD experiment, we use our first sample to make the gap between different curves large, and the size of this sample is 1,000.

5.4.2 Experimental Results and Experimental Analysis

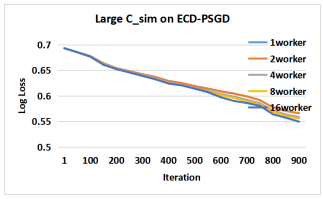


(a) Large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on minibatch SGD

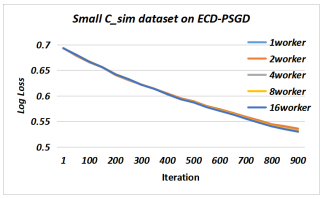


(b) Small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on minibatch SGD

Figure 7. The performance of different $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ datasets on minibatch SGD.

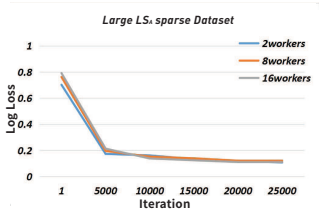


(a) Large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on ECD-PSGD

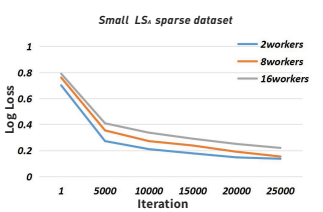


(b) small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on ECD-PSGD

Figure 8. The performance of different $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ datasets on ECD-PSGD.

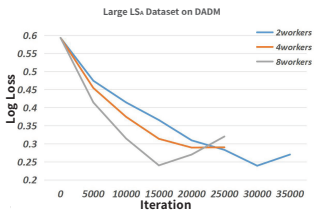


(a) Large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on Hogwild!

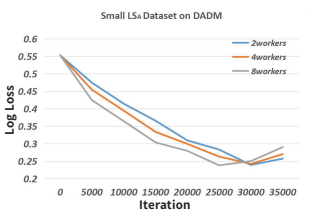


(b) Small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on Hogwild!

Figure 9. The performance of different $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ datasets on Hogwild!.In this case, the effect is better when the gap is small. The number of iterations for the server to reach a fixed ϵ is stable when increase the number of workers. Then, the number of iterations in each worker decreases.



(a) Large $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on DADM



(b) Small $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ dataset on DADM

Figure 10. The performance of different $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ datasets on DADM.

The experimental results are shown in Figures 7, 8,9 and 10. In our $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ experiment, our experimental results match

well with the theoretical analysis: a large $LS_{\mathcal{A}}(\mathcal{D}, \mathcal{S})$ value leads to better scalability. In minibatch SGD, DADM and ECD-PSGD, when $LS_{\mathcal{A}}(\mathcal{D}, \mathcal{S})$ is large, at the same iteration, the more gain-growth we can obtain: the gap between the different lines is large. For the ASGD algorithm, i.e., Hogwild!, when $LS_{\mathcal{A}}(\mathcal{D}, \mathcal{S})$ is large, we can obtain more gain-growth: the gap between the different lines is small, which means that each worker trains fewer iterations.

5.5 Speedup Upper Bound Experiment

5.5.1 Algorithm Setting

The algorithm setting in this section is the same as the feature variance and sparsity section. The above sections show that different datasets are applicable to different algorithms; we only present 1 sparse dataset for the Hogwild! feature variance dataset for minibatch SGD and ECD-PSGD.

Our experimental environment cannot reach the upper bound of speedup of the real-sim dataset: our experimental environment supports only twenty-four threads (workers) in all. Thus, in the Hogwild! experiment, we use a simulated dataset. In the minibatch SGD and ECD-PSGD experiments, we use the HIGGS dataset.

5.5.2 Experimental Results and Experimental Analysis

Table 2
The iteration per worker for different algorithms.

Algorithm	2 workers	4 workers	8 workers	16 workers	24 workers
Hogwild!	376	321	**356**	**412**	-
minibatch	91	87	**71**	**69**	-
ECD-PSGD	**1654**	**1621**	1623	1648	-
DADM	22596	11421	6258	**4064**	**3972**

The results are shown in Table 2. The upper bound is between two bold values. In Table 2, we show that the different algorithms have their upper bound speedup, which is highlighted in bold in Table 2, even using their best performance dataset. Based on our analysis in Section 4.2, the gain-growth for Hogwild! is negative. For ECD-PSGD and minibatch SGD, the growth is close to zero. Thus, in the range that we marked, the algorithms meet their speedup upper bounds.

6 Conclusion

Based on our analysis and experiments, we draw the following conclusion about $Speedup$ function's general mathematical properties and gain following conclusions: 1. Different datasets are applicable to different parallel stochastic optimization algorithms. 2. Before training a machine learning model, rearranging the dataset is an ideal choice. 3. Regardless of which parallel stochastic optimization algorithm is used, there always exists an upper bound of speedup. 4. The scalability performance for a certain dataset on a specific machine learning model cannot be applied to other cases.

It is worthy to note that some machine learning models, like CNN or DNN, do not obey the convex, Lipschitz, or continuity requirements. Thus, the scalability of the algorithm on these models is needed to be studied further.

References

[1] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," 2017.

[2] M. P. Perrone, H. Khan, C. Kim, A. Kyrillidis, J. Quinn, and V. Salapura, "Optimal mini-batch size selection for fast gradient descent," 2019.

[3] H. C. Cho, J. Kim, S. Kim, Y. H. Son, N. Lee, and S. H. Jung, "Passcode: Parallel asynchronous stochastic dual co-ordinate descent," *Neuroscience Letters*, vol. 519, no. 1, p. 78–83, 2015.

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 257–269, 2010.

[5] J. C. Duchi, A. Agarwal, and 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.

[6] L. Colin Igor and K. Scaman, "Theoretical limits of pipeline parallel optimization and application to distributed deep learning," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 12 350–12 359.

[7] J. Langford, A. J. Smola, and M. Zinkevich, "Slow learners are fast," in *International Conference on Neural Information Processing Systems*, 2009, pp. 2331–2339.

[8] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," *arXiv preprint arXiv:1804.03235*, 2018.

[9] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes." *arXiv: Computer Vision and Pattern Recognition*, 2017.

[10] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *Computer Science*, 2014.

[11] J. R. M. M. N. L. E. P. A. M. M. M. J. D. M. F. P. M. H. Thorsten Kurth, Sean Treichler, "Exascale deep learning for climate analytics," *SC '18 Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2018.

[12] Y. You, I. Gitman, and B. Ginsburg, "Scaling sgd batch size to 32k for imagenet training." *arXiv: Computer Vision and Pattern Recognition*, 2017.

[13] N. Dryden, N. Maruyama, T. Moon, T. Benson, M. Snir, and B. Van Essen, "Channel and filter parallelism for large-scale cnn training," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–20.

[14] Y. You, J. Hseu, C. Ying, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large-batch training for lstm and beyond," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–16.

[15] F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 24, pp. 693–701, 2011.

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

[17] S. Zheng, F. Xia, X. Wei, and Z. Tong, "A general distributed dual coordinate optimization framework for regularized loss minimization," *Journal of Machine Learning Research*, vol. 18, 2017.

[18] M. Abadi, "Tensorflow: learning functions at scale," *Acm Sigplan Notices*, vol. 51, no. 9, pp. 1–1, 2016.

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

[20] Y. You, J. Li, J. Hseu, X. Song, J. Demmel, and C. Hsieh, "Reducing bert pre-training time from 3 days to 76 minutes," *arXiv: Learning*, 2019.

[21] Y. M. Ermoliev, "On the stochastic quasi-gradient method and stochastic quasi-feyer sequences," 1969.

[22] ——, "On the stochastic quasi-gradient method and stochastic quasi-feyer sequences," 1969.

[23] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," in *Siam J Optim*, 2009, pp. 1574–1609.

[24] "On the stochastic quasi-gradient method and stochastic quasi-feyer sequences."

[25] L. Bottou, *Large-Scale Machine Learning with Stochastic Gradient Descent*. Physica-Verlag HD, 2010.

[26] TensorFlow, "Tensorflow: Google open sources their machine learning tool."

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM International Conference on Multimedia*, 2014, pp. 675–678.
[28] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice Hall, 2003.
[29] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 165–202, 2012.
[30] S. Li, T. Ben-Nun, S. D. Girolamo, D. Alistarh, and T. Hoefler, "Taming unbalanced training workloads in deep learning with partial collective operations," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 45–61.
[31] D. Cheng, S. Li, and Y. Zhang, "Wp-sgd: Weighted parallel sgd for distributed unbalanced-workload training system," *Journal of Parallel and Distributed Computing*, vol. 145, pp. 202–216, 2020.
[32] S. Li, T. Ben-Nun, D. Alistarh, S. Di Girolamo, N. Dryden, and T. Hoefler, "Breaking (global) barriers in parallel stochastic optimization with wait-avoiding group averaging," *arXiv preprint arXiv:2005.00124*, 2020.
[33] N. Feng, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 24, pp. 693–701, 2011.
[34] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent," 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. 2595–2603.
[35] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *Siam Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.
[36] S. Shalevshwartz and Z. Tong, "Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization," in *International Conference on International Conference on Machine Learning*, 2014.
[37] S. Shalevshwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *Journal of Machine Learning Research*, vol. 14, no. 1, p. 2013, 2013.
[38] M. Jaggi, V. Smith, M. Taká, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, "Communication-efficient distributed dual coordinate ascent," *Advances in Neural Information Processing Systems*, vol. 4, pp. 3068–3076, 2014.
[39] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Taká, "Adding vs. averaging in distributed primal-dual optimization," 2015.
[40] T. Yang, "Trading computation for communication: Distributed stochastic dual coordinate ascent," pp. 629–637, 2013.
[41] C. Ma, J. Konecný, M. Jaggi, V. Smith, M. I. Jordan, P. Richtarik, and M. Takac, "Distributed optimization with arbitrary local solvers," *Optimization Methods & Software*, vol. 32, no. 4, pp. 813–848, 2017.
[42]
[43] J. C. Duchi, F. Ruan, and C. Yun, "Minimax bounds on stochastic batched convex optimization," in *Conference on Learning Theory*.
[44] K. A. Y. Sra, "Sgd with shuffling: optimal rates without component convexity and large epoch requirements," *arXiv: Learning*, 2020.
[45] R. Leblond, F. Pedregosa, and S. Lacoste-Julien, "Improved asynchronous parallel optimization analysis for stochastic incremental methods," *Journal of Machine Learning Research*, vol. 19, 2018.
[46] L. L. Cam and G. L. Yang, "Asymptotics in statistics."

Daning Cheng received his Bachelor in computer science and technology in 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 optimzation 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 focus on parallel and distributed computing, and parallel and distributed deep learning.

Hanping Zhang born in 1994. Master candidate. His main research interests include automl, deep learning, and optimization algorithm.

Fen Xia is a chief scientist in the Beijing Wisdom Uranium Technology Co., Ltd. His research interests include automl machine learning, statistical machine learning, ranking, large scale machine learning algorithms, regularization methods, and information retrieval.

Yunquan Zhang received his BS degree in computer science and engineering from the Beijing Institute of Technology in 1995. He received a PhD degree in Computer Software and Theory from the Chinese Academy of Sciences in 2000. He is a full professor and PhD 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.

APPENDIX

6.1 Notes and Symbols

To make our present clearly, we summarize the algorithm descriptions common notes and symbols here. n is the number of sample in dataset. m is the number of worker. γ is the learning rate. λ is the regularization coefficient. $G_{\xi_i}(x)$ and $\nabla F(x;\xi_i)$ are the sub-gradient of function $F(x;\xi_i)$. To make reader easy to match the algorithm descriptions in their original paper, we keep them all in our algorithm descriptions. Q is the collection of samples which are in a mini-batch. $batch\_size$ is the number of Q and $local\_batch\_size$ is the number of Q_{local} which is the mini-batch in a worker.

To make our presentation clearly, in algorithm theory analysis parts, we omit non-relevant parameters in these following lemmas and theorems. In algorithm theory analysis parts, $h_i(\cdot)$ are the functions which only contains the parameters which are related to the machine learning model, initial value x_0 and algorithm parameter like λ and γ. $h_i(\cdot)$ do not care about the characteristic of datasets and how many nodes we will use, i.e. the value of m.

6.2 Hogwild!

6.2.1 Algorithm Description

Hogwild! is the most important asynchronous parallel SGD algorithm. Hogwild! is the base of current machine learning frame: Parameter Server framework.

The Algorithm 1 is the description of Hogwild!. It is worthy to mention that $F(x;\xi)$ is not the loss function directly. $F(x;\xi)$ should be written as hypergraph form [15].

Algorithm 1 Hogwild!

In: 1 Server, m worker, random delay τ ($0 < \tau < \tau_{max}$), learning rate γ
Out: x^*, which is the argmin of $f(x)$

WORKER:
repeat
1. Pick sample ξ_i from dataset;
2. Pull Model x_i from Server;
3. Compute $G_{\xi_i}(x_i)$, which is the sub-gradient of $F(x;\xi_i)$
4. Push $G_{\xi_i}(x)$ into Server.

until Forever

SERVER:
repeat
1. Receive $G_{\xi_i}(x_{j-\tau})$ from any worker.
2. $x_{j+1} = x_j + \gamma G_{\xi_i}(x_{j-\tau})$

until Forever
3. Return x^*

6.2.2 Theorem Analysis

Firstly, we present a necessary conclusion which builds the connection between the number of workers and the lag(delay) between when a gradient is computed and when it is used in Parameter Server Framework.

Theorem 1. The minimum of the maximum of τ is the number of workers, i.e. $m \leq \tau_{max}$. And when all workers share the same performance, the system would achieve the minimum.

Proof 1. In a M worker cluster, for slowest worker, at tth iteration, this slowest worker submit its gradient to server, At this time, other workers is in computing their gradient. at $t+j$th iteration, the slowest workers submit its gradient again. At this time, other workers is already submit at least one gradient in j iterations, i.e. $j > M$. Thus, an asynchronous parallel system at least has M iteration delay. And when all workers share the same performance, the system would achieve the minimum.

The convergence analysis of Hogwild! is shown in Theorem 2. This theorem is transformed theorem from the Niu et al. 's work [15].

Theorem 2. Suppose in Algorithm 1 that the lag, i.e. τ, which is between when a gradient is computed and when it is used, is always less than or equal to τ_{max}, and γ is under certain condition. for some $\epsilon > 0$.When t is an integer satisfying

$$t \geq (1 + 6\tau_{max}\rho + 6\tau_{max}^2\Omega\delta^{1/2})\Omega h(\epsilon)$$

Then after t component updates of x, we have $\mathbb{E}[f(x_t) - f(x^*)] < \epsilon$. $h(\epsilon)$ is only influenced by the characteristic of $f(\cdot)$ and initial value x_0.

In Theorem 2, ρ is the probability that any two $G_{\xi_i}(x_i)$ and $G_{\xi_j}(x_j)$ have the same nonzero value at the same feature; Ω is the max number of nonzero feature in $G_\xi(x)$; δ is simply the maximum frequency that any feature appears in $G_\xi(x)$.

Sparsity and Feature variance As we can see, when each worker shares the same performance, each worker needs to train $t/m = (1/m + 6\rho + 6m\Omega\delta^{1/2})\Omega h(\epsilon)$ which means with the increasing of the number of workers, each worker may have to exert more iterations. To make each workers training less iteration with increasing the number of workers, the $\Omega\delta^{1/2}$ should be extremely small: When m is large enough, we expect that $1/(m+1)+6(m+1)\Omega\delta^{1/2} < 1/m+6m\Omega\delta^{1/2}$, which means we can gain benefit when we use more resource, i.e. a good algorithm scalability. Above facts show that the scalability of Hogwild is controlled by the value $\Omega\delta^{1/2}$.

When we decide which machine learning model we use, the sparsity of dataset is the only factor which influences the Ω and δ. From the definition of Ω, δ and ρ, we can gain conclude that Ω, δ and the sparsity of $G_\xi(x)$ is a positive correlation. For common machine learning model, like SVM, LR, neural network, the relationship between the sparsity of samples in a dataset and the sparsity of $G_\xi(x)$ is clearly and significantly positive correlation. Especially, when machine learning models are linear models like SVM and LR, the sparsity of $G_{\xi_i}(x)$ is equal to the sparsity of ξ_i.

Above conclusion is also shown in other ASGD algorithms convergence analysis like delay-tolerate ASGD and quantization ASGD.

Theorem 2 shows that feature variance plays no influence on algorithm scalability. However, when the dataset is sparse, the feature variance must be low: for any feature, in most samples in the dataset, this feature is zero.

The influence of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ The influence of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is buried in the proof of Theorem 2. The conclusion is that $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is positively correlated to the scalability. The proof of this part we put in Appendix part for this part needs to cite a lot of proof context from the work [15].

In the Hogwild! proof, the τ is created in A8 and A6 parts in the appendix of the work [15]. In these equations, δ, Ω, ρ are created by the sum of multiplication of gradient G_{ξ_i} or model difference (x_i - $x_{k(i)}$, which can be described as G_{ξ_i}). The sum

range is ξ_i to $\xi_{i-\tau}$. Above facts show that the original definition δ, Ω, ρ is large: it is unnecessary to calculate these parameters in whole dataset. just it is better define these parameters server in sample sequence neighborhood τ_{max} samples sub-dataset: If we define δ, Ω, ρ as $\delta_{local}, \Omega_{local}, \rho_{local}$, which is calculated in sample sequence neighborhood τ_{max} samples sub-dataset and replace δ, Ω, ρ in Hogwild! proof, the whole proof of Hogwild! is still sound. So, we find a tighter upper bound of Hogwild! algorithm.

As we can see from the definition, when $LS_{\mathcal{A}}(\mathcal{D}, \mathcal{S})$ is small, $\delta_{local}, \Omega_{local}, \rho_{local}$ is also small, which would increase the scalability ability.

The upper bound of scalability From Theorem 2, we draw the scalability upper bound which is decided by the characteristic of dataset. To make time faster, at least each worker should train less sample compared with one worker, i.e. $1/m + 6m\Omega\delta^{1/2} < 1/1 + 6 * 1 * \Omega\delta^{1/2}$. However, the function $constant_1 x + constant_2/x$ $(constant_1, constant_2 > 0)$ is increasing function when x is large enough. Thus the maximum of m, which satisfies $1/m + 6m\Omega\delta^{1/2} < 1/1 + 6 * 1 * \Omega\delta^{1/2}$ is the maximum number of worker we can use in Hogwild!. The upper bound of Hogwild! scalability suits second situation in Section 4.2.

6.3 mini-batch SGD algorithm

6.3.1 Algorithm Description

Mini-batch SGD algorithm is the most critical data-parallel SGD algorithm. Nowadays, mini-batch SGD is the main parallel method which is implemented in the supercomputer.

Algorithm 2 is the description of mini-batch SGD algorithm.

Algorithm 2 Mini-batch SGD algorithm

In: 1 Server, $batch\_size$ Workers, learning rate γ
Out: x^*, which is the argmin of $f(x)$

WORKER:
for Forever **do**
1. Pick sample ξ_i from dataset;
2. Receive x_i from Server
3. Compute $G_{\xi_i}(x_i)$, which is the sub-gradient of $F(x_i; \xi_i)$
4.Push $G_{\xi_i}(x)$ into Server.
end for

SERVER:
for Forever **do**
1. All-gather $G_{\xi_1}(x_j), G_{\xi_2}(x_j), ..., G_{\xi_{batch\_size}}(x_j)$ from $worker_1, worker_2, , , , , worker_{batch\_size}$
2. Compute $G_{ave}(x_j) = \frac{1}{batch\_size} \sum_{i=1}^{batch\_size} G_{\xi_i}(x_j)$
3. $x_{j+1} = x_j + \gamma G_{ave}(x_j)$
end for
4. Return x^*

6.3.2 Theorem Analysis

Again, we present the basic fact which builds the connection with the degree of parallelism and batch size. The following fact is valid.

Fact 1. In Algorithm 2, the upper bound of the number of workers is the batch size.

To make our presentation clear, we show our theorem about the convergence of the mini-batch SGD algorithm:

Theorem 3. When objective function Eq. 2 is running on Algorithm 2, then we have

$$\mathbb{E}_{x_t \in D_t} f(x_t) - f(x^*) \leq \left(\left(h_2\left(F\left(\cdot\right)\right) \left(d(\mu_{D^t}, \mu_{D^*}) + \frac{\sigma_{D^*} + Was_2(D^0, D^*)(1-\gamma\lambda)^t}{(batch\_size)^{t/2}} \right) + h_3\left(F\left(\cdot\right)\right) \right)^2$$

where D^t is the distribution of x^t, D^* is the distribution of x^*, σ_D is the standard deviation of distribution D, μ_D is the mean of distribution D. $Was_2(D_1, D_2)$ is the Wasserstein metrics between D_1 and D_2.

Proof 2. Based on the work by M.Zinkevich et al [34], we treat x_t as random variable firstly and its distribution is D_t. We have following theorem (Theorem 11 in M.Zinkevich [34]) Given a cost function f such that $\|f\|_L$ and$\|\nabla f\|_L$($\|\cdot\|_L$ is Lipschitz seminorm) are bounded, a distribution D such that σ_D and is bounded , then ,for any v

$$\begin{aligned} &\mathbb{E}_{x \in D}[f(x)] - \min_x f(x) \leq \\ &(W_2(v, D)) \sqrt{2 \|\nabla f\|_L \left(f(v) - \min_x f(x)\right)} \\ &+ \|\nabla f\|_L \left(W_2(v, D)\right)^2 / 2 + \left(f(v) - \min_x f(x)\right) \end{aligned} \tag{5}$$

When $v = \mu_{D^*}$ $W_2(\mu_{D^*}, D)$ is the relative standard deviation of x_t with respect to μ_{D^*}, i.e. $\sigma_D^{\mu_{D^*}}$.
Based on Theorem 32 in M.Zinkevich et al [34], we know that

$$\sigma_D^{\mu_{D^*}} \leq \sigma_D + d(\mu_{D^*}, \mu_D) \tag{6}$$

$$\sigma_{D^t} \leq \sigma_{D^*} + W(D^*, D^0)(1 - \lambda\gamma)^t \tag{7}$$

Suppose that random variable $X^1, X^2, X^3, ..., X^k$ are independent and identically distributed. if $A = \frac{1}{k}\sum_{i=1}^{k} X^i$, it is the case that:

$$\mu_A = \mu_{X^1} = \mu_{X^2} = ... = \mu_{X^k}$$
$$\sigma_A \leq \frac{\sigma_{X^1}}{\sqrt{k}}$$

As we can see from x_t, before average operation, x_t^i is independent and identically distributed random variable. In each iteration, σ_{D^t} is shrinked $\frac{1}{batch\_size}$. Combining above equations, we can get theorem.

Sparsity and Feature variance When dataset and machine learning model are chosen, D^0 and D^* would be determined. For most of the cases, the x^* is a fixed number. The value of $Was_2(D^0, D^*)$ is determined by the characteristic of D^0: Based on the definition of Wasserstein metrics, we can know that $Was_2(D^0, D^*)$ is positive correlative to the variance of D^0. It is evident that when a machine learning model is determined, sample variance is positively correlated with the variance of D^0. Thus, when sample variance is significant, the gain, which is brought by parallel, is remarkable.

The feature variance is positively correlated with sample variance. Thus, the dataset with higher feature variance is suited to mini-batch SGD. Although the Theorem 3 do not show the effect of the sample sparsity, yet we know that the feature variance is negatively correlated to sample sparsity. Thus, sparse datasets do not suit mini-batch SGD.

The influence of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ In this algorithm, the sample sequence we discuss is the sequence which build by the sample batch and we pick the sequence which can build the maximum $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$. For example, in Algorithm 2, batch_size is 3 and the sequence of server received $G_\xi(x)$ is $\{G_{\xi_1}(x_1), G_{\xi_2}(x_1), G_{\xi_3}(x_1)\}$,$\{G_{\xi_4}(x_2), G_{\xi_5}(x_2), G_{\xi_6}(x_2)\}$,..., $\{G_{\xi_{3t-2}}(x_t), G_{\xi_{3t-1}}\xi_2(x_t), G_{\xi_{3t}}(x_t)\}$, where the sample or gradient in $\{\cdot,\cdot,\cdot\}$ is in on batch. Then, the $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ for mini-batch SGD algorithm is the $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ for $\xi_i, \xi_{i+1}, \xi_{i+2}$ and $\xi_i, \xi_{i+1}, \xi_{i+2}$ can build a sequence whose $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is the maximum in all batches.

$LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small means that, at every iteration, most feature do not gain more information from a batch, i.e. mini-batch SGD is invalid at the most feature in every iteration. Above fact suggest that when $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small, the parallel effect is poor.

The upper bound of scalability As we can see from Theorem 2, the gain at t-th iteration offered by parallel is $\frac{1}{(batch\_size)^t}$, which means that the gain growth is decreasing with the increasing of batch_size. Although in theory, enlarging batch_size always gains more profit, yet the gain growths are small when batch_size is large enough. When the gains cannot cover the parallel cost, the scalability reaches its upper bound. The upper bound of mini-batch SGD scalability suits first situation in Section 4.2.

6.4 DADM

6.4.1 Algorithm Description

DADM [17] depends on dual ascent method to gain a minimum of $f(x)$. The DADM can be treated as the mini-batched SDCA algorithm. DADM selected an intermediate variable to help different components of mini-batch are computed in the different node in a cluster.

The full version of DADM can be complex, and it tries to solve the objective function which contains three parts. However, when it comes to the common machine learning problem, the algorithm is presented in a simple form, like Algorithm 3. In Algorithm 3, $L(x;\xi)$ is the loss function. L^* and ψ^* is the convex conjugate function of F and ψ. α_i is the dual variables. To make our present clearly, we omit some explanations. Some notes are different with the original algorithm description [17]. Again, in this paper, our target is not showing every detail of the algorithm. We focus on algorithm scalability performance.

6.4.2 Theorem Analysis

The parallel influence on parallel stochastic gradient algorithm is reflected in the parameters in the theorem. However, DADM uses different proof structure to offer the convergence conclusion. In the proof of DADM: different workers solve a local problem, i.e. $f_i(x)$ in Eq. 10 at each iteration and then broadcast its information to other workers to solve globe problem $f(x)$ in Eq. 9. What is more, DADM is to find the expected duality gap. Thus, the convergence analysis conclusion is unrelated to a dataset and machine learning model character, and the parallel influence is buried in the problem setting instead of directly convergence theorem. The convergence theorem about DADM is the conclusion from the work [17].

Theorem 4. $f(\cdot)$, ξ_i and $\Delta\alpha_{local}$ satisfy some requirements. When t satisfies following condition, the expected duality gap of objective function and its dual form is smaller than ϵ

$$t \geq (h_5(\xi_i,\gamma,\lambda) + \frac{1}{local\_batch\_size * m})log\Big((h_5(\xi_i,\gamma,\lambda) + \frac{1}{local\_batch\_size * m})h_6(x_0,\epsilon)\Big)$$

Algorithm 3 DADM

In: 1 Server, m Workers, $batch\_size = n * local\_batch\_size$, learning rate γ, α_i=v^0=0
Out: x^*, which is the argmin of $f(x)$

WORKER:
for t=1,2,...,forever **do**
1. Pick $local\_batch\_size$ samples as Q_{local}, ξ_{j_1}, ξ_{j_2} ... $\xi_{j_{local\_batch\_size}} \in Q_{local}$ from dataset;
2. Receive Δv^{t-1} from Server
3. $v^t_{local} = v^{t-1}_{local} + \Delta v$
4. Approximately maximize Eq. 8,w.r.t $\Delta\alpha_i$

$$\Delta\alpha_{Q_{local}} = \underset{\alpha_{Q_{local}}}{argmin} \sum_{i\in Q_{local}} -L^*(-\alpha_i^{t-1} - \Delta\alpha_i) - \lambda\psi^*(v^{t-1}_{local} + \frac{\sum_{i\in Q_{local}} \xi_i \cdot \Delta\alpha_i}{\lambda n/m}) \quad (8)$$

5. Send $\Delta v^t_{local} = \frac{1}{\lambda}\sum_{i\in Q_{local}} \xi_i \cdot \Delta\alpha_i$
end for

SERVER:
for t=1,2,...,Forever **do**
1. All-gather $\Delta v^t_{local\_i}$ from $worker_i (i = 1, 2..., m)$
2. Compute $\Delta v^t = \frac{1}{n}\sum_{i=1}^m \Delta v^t_{local\_i}$
3. Broadcast Δv^t to all workers
end for
4. Return $x^* = \nabla\psi^*(v)$

Sample Diversity As we can see from the proof, the primary purpose of parallel technology is to cut the original problem into several subproblems. Thus, from the aspect of subproblem, the parallel algorithm will fail to accelerate the algorithm when some nodes solve the same problem. To ensure different nodes solve different subproblem, we should ensure dataset is high sample diversity. For example, when a dataset consists of little kinds of the sample, i.e. the dataset is the replication of a little sample, the sub-dataset in each node in the cluster would be almost the same, which means $f_i(x), \forall i$ in Eq. 10 are the same. In this case, DADM fails to make full use of multi-nodes. Thus, we can know that DADM is apt to accelerate the dataset whose sample diversity is high.

The influence of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ The influence of similarity is hard to be shown in theory analysis. However, from Algorithm 3 description step 2 in SERVER part, we can observe that using the definition of $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ in mini-batch SGD, when $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small, v^t_{local}s from a different worker would be almost the same, which would decrease the influence of parallel. Above fact suggest that when $LS_{\mathcal{A}}(\mathcal{D},\mathcal{S})$ is small, the parallel effect is poor.

The upper bound of scalability Again, the upper bound of scalability for DADM shares the same characteristics with mini-batch SGD. As the mini-batch SGD, the profit offered by parallel is $1/m$, which means that the gain growth is decreasing with

the increasing of m. Although in theory, enlarging m always gains more profit, yet the gain growths are small when m is large enough. When the gains cannot cover the parallel cost, the scalability reaches its upper bound. The upper bound of DADM scalability suits first situation in Section 4.2.

6.5 ECD-PSGD

6.5.1 Algorithm Description

Decentralization and compression stochastic gradient methods are a new hot topic. To reduce the burden of the network, different workers send compressed information to neighbourhood workers. Then, they average their models.

We choose one of the states of the art decentralization and quantization SGD algorithm: ECD-PSGD [16] as our example. In ECP-PSGD, we will show how datasets influence the algorithm scalability.

The description of ECP-PSGD is shown in Algorithm 4. Again, we still omit some explanations. We only offer a basic version of ECD-PSGD algorithm: all nodes share the same amount of data, and all nodes share the same weight. In this algorithm description, $x^{(i)}$ is the model in ith worker. The worker weight and network are described by matrix W. $W_{i,j}$ is the element in W's i row and j column and $1 = \sum_{i=1}^{m} W_{i,j} = 1$. The connected neighbours of one worker i here refers to all workers that satisfy $W_{i,j} \neq 0$.

Algorithm 4 ECD-PSGD

In: m Workers, Weighted and network matrix M_W, learning rate γ, initial point $x_1^i = x_0$, initial intermediate variable $y^{(i)} = x_0$

Out: x^*, which is the argmin of $f(x)$

WORKER:

for t=1,2,...,forever **do**

1. Pick a sample ξ_t from dataset;

2. Compute a local stochastic gradient based on ξ_i: $\nabla F(x_t^{(i)}; \xi_t)$

3. Pull compressed $y^{(j)}$ as $\hat{y}^{(j)}$ from neighbors worker and compute

$$x_{t+\frac{1}{2}} = \sum_{j=1}^{m} M_{W\,i,j} \hat{y}_t^{(j)}$$

Update local model

$$x_{t+1} = x_{t+\frac{1}{2}} - \gamma \Delta F(x_t^{(i)}; \xi_t)$$

4. Each worker compute the z-value of itself:

$$z_{t+1}^{(i)} = (1 - t/2) x_t^{(i)} + \frac{t}{2} x_{t+1}^{(i)}$$

and compress $z_{t+1}^{(i)}$ into $C(z_{t+1}^{(i)})$

5. Each worker update intermediate variable for its connected neighbors:

$$y_{t+1}^{(i)} = (1 - 2/y) y_t^{(i)} + \frac{2}{t} C(z_{t+1}^{(i)})$$

end for

6. Output:$x^* = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$

6.5.2 Theorem Analysis

To present the convergence analysis of Algorithm 4, we have to rewrite the objective function in to following form.

$$\begin{aligned} f(x) &= \frac{1}{n} \sum_{i=1}^{n} F(x; \xi_i) \\ &= \frac{1}{m} \sum_{j=1}^{m} \frac{1}{n_{local}} \sum_{i=0}^{n_{local}} F(x; \xi_{i,j}) \end{aligned} \tag{9}$$

And we also define following notes:

$$f_i(x) := \frac{1}{n_{local}} \sum_{i=0}^{n_{local}} F(x; \xi_{i,j}) \tag{10}$$

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x^(i)$$

$$\hat{\sigma}^2 \geq \frac{1}{n_{local}} \sum_{j=1}^{n_{local}} \left\| \nabla F(x; \xi_j) - \nabla f_i(x) \right\|^2, \forall x$$

$$\zeta^2 \geq \frac{1}{m} \sum_{i=1}^{m} \left\| \nabla f_i(x) - f(x) \right\|^2, \forall x$$

$$\mathbb{E}\left(C(z_t^{(i)}) - z_t^{(i)} \right) = 0, \forall x, \forall t, \forall i$$

$$\widetilde{\sigma}^2 \geq 2\mathbb{E} \left\| C(z_t^{(i)}) - z_t^{(i)} \right\|^2, \forall x, \forall t, \forall i$$

For Algorithm 4, Hanlin T et al. [16] gives following convergence theorem.

Theorem 5. In Algorithm 4, choosing an appropriate γ, it admits

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E} \left\| \nabla f(\overline{x}) \right\| \leq \frac{\hat{\sigma}}{\sqrt{mT}} + \frac{\hat{\sigma} \widetilde{\sigma}^2 log T}{m\sqrt{mT}} + \frac{\zeta^{2/3} \widetilde{\sigma}^2 log T}{mT\sqrt{T}} + h_4(\widetilde{\sigma}, \zeta, T) \tag{11}$$

As we can see from Algorithm 4, ECP-PSGD can be treated as the variant of mini-batch SGD: When the network W is fully connected, $x = C(x), t \to \inf$, ECD-PSGD degenerates into mini-batch SGD. Thus, ECD-PSGD inherits the characteristic of mini-batch SGD.

Sparsity and Feature variance Following mini-batch SGD, ECD-PSGD is apt to accelerate the dataset whose variance is large (and the dataset is dense). What is more, the m is also related to $\widetilde{\sigma}$, which means the ECD-PSGD is apt to accelerate the dataset, which would lose their a lot accurate during compress process.

The influence of $LS_{\mathcal{A}}(\mathcal{D}, \mathcal{S})$ The influence of similarity is the same with mini-batch SGD.

The upper bound of scalability Again, the upper bound of scalability for ECD-PSGD shares the same characteristics with mini-batch SGD. As the mini-batch SGD, the profit offered by parallel is $1/\sqrt{m}$, which means that the gain growth is decreasing with the increasing of m. Although in theory, enlarging m always gains more profit, yet the gain growths are small when m is large enough. When the gains cannot cover the parallel cost, the scalability reaches its upper bound. The upper bound of ECD-PSGD scalability suits first situation in Section 4.2.