The Scalability for Parallel Machine Learning Training Algorithm: Dataset Matters

Daning Cheng∗†, Hanping Zhang†§, Fen Xia‡, Shigang Li¶ and Yunquan Zhang∗
†University of Chinese Academy of Sciences, Beijing, China
‡SKL of Computer Architecture, Institute of Computing Technology, CAS, China
∗SKL of Computer Architecture, Institute of Computing Technology, CAS, China
Email: {chengdaning.zyq}@ict.ac.cn
¶Department of Computer Science, ETH Zurich, Switzerland
Email: shigangli.cs@gmail.com
§University at Buffalo, State University of New York
†Wisdom Uranium technology Co.Ltd, Beijing
{ Xiafen, Zhanghanping }@ebrain.ai

Abstract—To gain a better performance, many researchers put more computing resource into an application. However, in the AI area, there is still a lack of a successful large-scale machine learning training application: The scalability and performance reproducibility of parallel machine learning training algorithm are limited and there are a few pieces of research focusing on why these indexes are limited but there are very few research efforts explaining the reasons in essence.

In this paper, we propose that the sample difference in dataset plays a more prominent role in parallel machine learning algorithm scalability. Dataset characters can measure sample difference. These characters include the variance of the sample in a dataset, sparsity, sample diversity and similarity in sampling sequence.

To match our proposal, we choose four kinds of parallel machine learning training algorithms as our research objects: (1) Asynchronous parallel SGD algorithm (Hogwild! algorithm) (2) Parallel model average SGD algorithm (Mini-batch SGD algorithm) (3) Decenterilization optimization algorithm (ECD-PSGD) and (4) Dual Coordinate Optimization (DADM algorithm). These algorithms cover different types of machine learning optimization algorithms.

We present the analysis of their convergence proof and design experiments. Our results show that the characters datasets decide the scalability of the machine learning algorithm. What is more, there is an upper bound of parallel scalability for machine learning algorithms.

Index Terms—Parallel algorithm, Scalability, Dataset, Optimization method

I. INTRODUCTION

Training a machine learning model is an exhausting job. Training a machine model often uses (stochastic) optimization method, like (stochastic) gradient descent, Newton method or dual coordinate ascent method. With the development of parallel computing methods, to reduce training time, parallel and distribution optimization methods are proposed. Nowadays, machine learning frameworks, which use these distribution optimization methods, are widely used in AI and machine learning industry like MXNet, Tensorflow.

However, the scalability for those machine learning frame and algorithm is limited:

1. Although researchers offered state of the art large-scale machine learning training applications, those applications do not run machine learning training process on large-scale parallel system. (1) Some works focus on training specific machine learning model on a particular dataset. For example, some researchers use a specific DNN training Imagenet dataset, and their work cannot be pushed into other machine learning model and dataset. (2) Some works try their best to optimization math kernel, like matrix multiplication(GEMM kernel), for machine learning.

2. For general cases, with the more parallel computing resource throwing into those machine learning frame, it is evident that the effect of those frameworks does not improve too much. Some works claim that current distribution machine learning frameworks can only contain less 100 nodes.

Current parallel machine learning works are unsatisfied: (1) The improvement of distributed parallel machine learning is small with a large parallel computing resource. In some cases, the influence of using a large parallel computing resource can be harmful. (2) Many works are lack of replicability. The scalability performance for specific machine learning model on the specific dataset cannot be pushed into other models or datasets.

Thus, we proposed the question that: are the current state of the art parallel optimization methods able to run on super large-scale parallel computing environment? Besides algorithm design and engineering implements, are there any other factors which play critical roles on the scalability for parallel machine learning training algorithm?

To solve the above questions, in this paper, we choose four different kinds of state of the art parallel optimization methods as our benchmarks: (1) Asynchronous parallel SGD algorithm, ASGD (Hogwild! algorithm), (2) Parallel model average SGD algorithm (Mini-batch SGD algorithm), (3) Decenterilization optimization algorithm (ECD-PSGD) and (4) Dual Coordinate Optimization (DADM).

After examining the convergence analysis of the above algorithms, we find the sample difference plays vital roles in the scalability. Some characters of dataset can describe sample difference. Those characters include(1) the variance of sample
feature in a dataset. (2) the sparsity of sample in a dataset. (3) the diversity of the sample in a dataset. (4) the similarity of two successive sampling sample.

What is more, we also find that for most the algorithms, the gain growth is minor with the increasing of using the parallel resource in mathematic. Above fact shows that for most of the training algorithm, there is the scalability upper bound.

To prove our analysis, we conduct experiments. We design and choose different datasets which share different characters. Our experiment results match the convergence proof analysis.

Our contribution is summarized as follow:

1. We examined four different state-of-the-art parallel optimization methods. In the view of convergence analysis, we show that the characters of dataset play crucial roles in parallel machine learning algorithms scalability. The characters at least include (1) the variance of sample feature in a dataset. (2) the sparsity of sample in a dataset. (3) The diversity of the sample in a dataset. (4) the similarity of two successive sampling sample.

2. Different datasets suit different optimization methods.

3. The scalability of optimization algorithm has its upper bound which is decided by the dataset.

4. We design experiments to prove the importance of dataset on parallel machine learning algorithms scalability. We also show the upper bound of algorithm scalability on experiment datasets.

Our analysis and experimental results answer the following problem:

1. The current parallel machine learning algorithms cannot make full use of large-scale parallel computing environment, like a supercomputer. These large parallel computing environments’ parallel degree is much higher than the upper bound of algorithm scalability.

2. One scalability performance of an algorithm on a specific dataset cannot be pushed into other datasets.

3. To improve scalability, pre-processing is also necessary like the random sort for datasets.

II. RELATED WORKS

With the development of parallel computing and optimization methods, many parallel optimization methods are designed to make machine learning training process fast. The most widely used methods are different parallel SGD algorithm, and the newest state of the art methods include decentralization algorithm and dual optimization algorithms.

A. Parallel SGD algorithm

SGD can be dated back to the early work of Robbins and Monro [9], [10], [11], [12], [13]. Recent years, combining with the GPU and clusters [14], [15], parallelized SGD became the most powerful weapon solving machine learning problems [16], [17], [18], [19]. Parallel SGD can be roughly classified into 2 catagories - Asynchronous parallel SGD and Model Average Parallel SGD.

The goal results for Asynchronous Parallel SGD algorithm and sequential SGD are the same in fixed iterations. Model Average Parallel SGD algorithms gives us the answer about how to calculate a better output in fixed number of iterations.

1) Asynchronous Parallel SGD algorithm: Asynchronous Parallel SGD algorithm may first appear in J. Langford et al’s work [19]. When using the Parameter Server Pattern, instead of the gradient of the newest models, the workers compute the gradient from the models which are older than the newest model in \( \tau \) iterations, but not exceed a limit \( \tau < \tau_{max} \).

In Hogwild! Algorithm [20], under some restrictions like smooth and separable, parallel SGD can be implemented in a lock-free style, which is robust to noise [21]. However, these two methods lead to that the bound of the regret of the delayed-update algorithm will increase by \( o(\tau^2) \). To ensure the delay is limited, the communication overhead is unavoidable, which would hurts performance. In this paper, we call these kinds of algorithms as delaySGD. In engineering aspect, those kinds of algorithm often present in parameter server, and the implement of parameter server includes ps-lite in MXNET [22], TensorFlow [14], petuum [23]. The methods that constrict the delay is offered by Ho et al [24]. However, if the workers have different performance, the \( \tau \) would be enlarged, which means the convergence speed would be slow.

2) Model Average Parallel SGD algorithm: Asynchronous Parallel SGD algorithms can be considered as accelerating process of sequential SGD. Model Average Parallel SGD algorithm use different ways to compute more accurate result. Martin.Z [25] gives us a parallel SGD method, which has almost no communication overhead. Zhang Y [26] gives a sharp analysis and proof of this parallel way. However, these methods do not take heterogeneous computing environment into account. Zhang Y et al [26] also points out that the SimuParal SGD does not always work. In fact, in our view, the effect of SimuParal SGD mainly depends on how large the model’s relative standard deviation is. However, not all training datasets have ability to make models have large relative standard deviation. In engineering aspect, these kinds of algorithms is kind of MapReduce algorithm [27]. So, most of them are Running on platforms like Spark [28] or Hadoop [29].

B. Decentralization SGD algorithm and Quantization

Decentralization algorithm, which is also named gossip algorithms [30], is another algorithm design trend. Decentralization algorithms are used to solve the consensus, where the network topology is decentralized problems [31]-[33]. Recent works show decentralization algorithms could outperform the centralized counterpart algorithm because decentralization algorithms reduce the network hot spot [31].

Decentralized parallel stochastic gradient descent [35] is one kind of the decentralization algorithm. Decentralized parallel stochastic gradient descent requires each node to exchange its own stochastic gradient and update the parameter using the information it receives [7].

With the unbiased stochastic compression technology, whose popular name is quantization, decentralization algorithms further display their superiority on network burden.
Current methods include randomized quantization [33], randomized sparsification [36] and other technology [37] [38] [39].

C. Dual Coordinate Ascent Optimization

Stochastic dual coordinate ascent method(SDCA) [40] [41] is one of the most important optimization method. Its data parallelism algorithms are hot topic in optimization algorithm area [42] [43].

DADM [8], DisDCA [44], CoCoA+ [45] are state of the art distribution parallel Dual Coordinate Optimization.

III. BACKGROUND

A. Problem Setting

For machine learning, an optimization method is used to solve the following minimum problem:

\[ \min \hat{f}(x) = \mathbb{E}_\xi F(x; \xi) \]

where \( \xi \) is a random variable which satisfies a certain distribution. For the most cases, the distribution of \( \xi \) is unknown or cannot be presented as a formula form. It is common that we use frequency histogram to replace PDF. Above formula is written as:

\[ \min \hat{f}(x) = \mathbb{E}_\xi F(x; \xi) \approx f(x) = \frac{1}{n} \sum_{i=1}^{n} F(x; \xi_i) \]

where \( \xi_i \) is the sample which sampled from \( \xi \). The collection of \( \{\xi_1, \xi_2, \ldots, \xi_n\} \) is the dataset. And \( x^* = \arg\min f(x) \).

For regularized risk minimization, \( F(x; \xi) \) is usually presented as following formula [23]:

\[ F(x; \xi) = L(\xi, x) + \lambda \psi(x) \]

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

\[ F(x; \xi) = L(\xi, x) + \frac{\lambda}{2} ||x||^2 \] (1)

B. 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. \( \gamma \) is the learning rate. \( \lambda \) is the regularization coefficient. \( G(\xi, x) \) and \( \nabla F(x; \xi) \) are the sub-gradient of function \( F(x; \xi) \). 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.

C. Hogwild!

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[5].

Algorithm 1 Hogwild!

In: 1 Server, \( m \) worker, random delay \( \tau \) ( \( 0 < \tau < \tau_{max} \)), learning rate \( \gamma \)

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

WORKER:
repeat
1. Pick sample \( \xi_i \) 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 \( x^* \)

D. Mini-batch SGD algorithm

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 \( \gamma \)

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

WORKER:
for Forever do
1. Pick sample \( \xi_i \) from dataset;
2. Receive \( x_i \) 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.
end for

SERVER:
for Forever do
1. All-gather \( G_{\xi_1}(x_1), G_{\xi_2}(x_2), \ldots, G_{\xi_{batch\_size}}(x_{j}) \) from worker1, worker2, ..., 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 \( x^* \)

E. Distributed Alternating Dual Maximization(DADM)

DADM [8] 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 goal 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 \( \psi^* \) is the convex conjugate function of \( F \) and \( \psi \). \( \alpha_l \) is the dual variables. To make our present clearly, we omit some explanations. Some notes are different with the original algorithm description [8]. Again, in this paper, our target is not showing every detail of the algorithm. We focus on algorithm scalability performance.

**Algorithm 3 DADM**

**In:** 1 Server, \( m \) Workers, \( \text{batch} \_\text{size} = n * \text{local} \_\text{batch} \_\text{size} \), learning rate \( \gamma \), \( \alpha_l = 0 \)

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

**WORKER:**

for \( t=1,2,...,\text{forever} \) do

1. Pick \( \text{local} \_\text{batch} \_\text{size} \) samples as \( Q_{\text{local}}, \xi_{j1}, \xi_{j2} \ldots \)
2. Compute \( F_{\text{local}} = \sum_{i \in Q_{\text{local}}} \hat{y}_{ij}^2 \) from dataset;
3. Receive \( \Delta v^t \) from Server
4. Approximately maximize Eq. 2 w.r.t \( \Delta \alpha_i \)
   \[ \Delta \alpha_{Q_{\text{local}}} = \text{argmin}_{\alpha_{Q_{\text{local}}} \in Q_{\text{local}}} \sum_{i \in Q_{\text{local}}} -L^*(-\alpha_{i}^{t-1} - \Delta \alpha_i) - \lambda \psi^*(\Delta v^t) \]
5. Send \( \Delta v_{\text{local}}^t = \frac{1}{m} \sum_{i \in Q_{\text{local}}} \xi_i \cdot \Delta \alpha_i \)

**SERVER:**

for \( t=1,2,...,\text{forever} \) do

1. All-gather \( \Delta v_{\text{local}}^t \) from worker \( i = 1, 2, ..., m \)
2. Compute \( \Delta v^t = \sum_{i=1}^{m} \Delta v_{\text{local}}^t_j \)
3. Broadcast \( \Delta v^t \) to all workers
4. Return \( x^* = \nabla \psi^*(v) \)

**F. ECD-PSGD**

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

We choose one of the states of the art decentralization and quantization SGD algorithm: ECP-PSGD [7] 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 ECP-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 \( i \)th worker. The worker weight and network are described by matrix \( W \). \( W_{ij} \) is the element in \( W \)’s \( i \) row and \( j \) column and \( I = \sum_{m=1}^{n} W_{ij} = 1 \). The connected neighbours of one worker \( i \) here refers to all workers that satisfy \( W_{ij} \neq 0 \).

**Algorithm 4 ECD-PSGD**

**In:** \( m \) Workers, Weighted and network matrix \( W \), learning rate \( \gamma \), initial point \( x^{(1)} = x_0 \), initial intermediate variable \( y^{(i)} = x_0 \)

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

**WORKER:**

for \( t=1,2,...,\text{forever} \) do

1. Pick a sample \( \xi_t \) from dataset;
2. Compute a local stochastic gradient based on \( \xi_t \): \( \nabla F(x^{(i)}_t; \xi_t) \)
3. Pull compressed \( y^{(j)} \) as \( \hat{y}^{(j)} \) from neighbors worker and compute
   \[ x^{(i+1)}_t = \sum_{j=1}^{m} W_{ij} \hat{y}^{(j)}_t \]
4. Update local model
   \[ x^{(i+1)} = x^{(i)} - \gamma \Delta F(x^{(i)}; \xi_t) \]
5. Each worker compute the z-value of itself:
   \[ z^{(i)}_t = 1 - \frac{t}{2} x^{(i)} - \frac{t}{2} \Delta F(x^{(i)}; \xi_t) \]
6. And compress \( z^{(i)}_t \) into \( C(z^{(i)}_t) \)
7. Each worker update intermediate variable for its connected neighbors:
   \[ y^{(i)}_t = (1 - \frac{t}{2} y^{(i)}_t) + \frac{t}{2} C(z^{(i)}_t) \]

**SERVER:**

for \( t=1,2,...,\text{forever} \) do

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

IV. THE UPPER BOUND OF SCALABILITY

A. The index to measure sample difference

1) The similarity of consecutive samples in the sampling sequence: To make our presentation clearly, we have to define the similarity of consecutive samples in the sampling sequence.

For a sampling sequence \( \xi_1, \xi_2, ..., \xi_n \) and a range \( \text{range} \), the similarity of consecutive samples in the sampling sequence is defined as

\[ C_{\text{sim}\_\text{range}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{\text{range}} \| \xi_i - \xi_{i+j} \|_0 \]

where \( \text{length} \) is the sequence length.

We set this index because online learning applications often use SGD as their optimization method. In an online learning
application, the samples in the sample sequence are often similar to its neighbourhood samples. For example, the online sample from advertisement click is similar to its neighbourhood, because user interest cannot be changed drastically.

In our following analysis, we would conclude that break similarity would gain better scalability. $C_{\text{sim}}$ is the parameter which measures similarity.

For a sample collection $\{\xi_1, \xi_2, \ldots, \xi_n\}$, their different sampling orders have different $C_{\text{sim}}$.

Example 1: For dataset $(0,0,0), (0,0,1), (0,1,1), (0,1,0), (1,1,0), (1,0,0)$, the samples have 2 different $C_{\text{sim}}$ sequence:
1. Sequence with $C_{\text{sim}2}=0.5$: $(0,0,0), (0,0,1), (0,1,1), (0,1,0), (1,1,0), (1,0,0)$
2. Sequence with $C_{\text{sim}2}=1$: $(0,0,0), (1,1,0), (0,0,1), (1,0,0), (0,1,0), (0,1,1)$

2) Feature variance and sparsity: In this paper, we define the variance of feature $k$ as

$$
\text{feature mean}_k = \frac{1}{n} \sum_{i=1}^{n} \xi_i(k)
$$

$$
\text{feature variance}_k = \frac{1}{n} \sum_{i=1}^{n} (\xi_i(k) - \text{feature mean}_k)^2
$$

where $\xi_i(k)$ is the $k$-th feature in $\xi_i$.

The sparsity is the rate between the number of zero elements with the size of sample.

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

3) Diversity: The diversity is the number of different kinds of samples in dataset. We notice that the size of 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 index to describe the sample difference. In following example, we will show that low variance, low density dataset still can have high diversity.

Example 2: Low density dataset whose sample size is large and diversity is high: $(1,0,0,...,0), (0,1,0,...,0), \ldots, (0,0,0,...,1)$. Diversity cannot be present by variance and sparsity. Thus, it is necessary to use this index to describe the sample difference. In following example, we will show that low variance, low density dataset still can have high diversity.

Example 3: The diversity of low variance dataset $(0.01),(0.02),(0.03),\ldots,(0.99),(1)$ is higher than the diversity of high variance dataset $(100),(-100),(100),(-100),\ldots,(100),(-100)$.

B. Perfect Computer Assumption

To avoid the discuss of the code implementation, parallel math kernel implementation and hardware setting, we assume that the nodes in a cluster have unlimited memory and the bandwidth in the network. Under this assumption, we can focus on the degree of parallelism, which is offered by the algorithm.

C. The Upper Bound of Algorithm Scalability

Gain, Cost and Gain Growth For ASGD algorithm like Hogwild!, the cost is the number of iterations for each worker. For DADM, ECD-PSGD and mini-batch SGD, the gain is the value of goal function at a fixed iteration.

The gain growth is the value of goal function’s difference or the cost difference between using $m$ nodes and $m+1$ nodes at a fixed iteration:

For ASGD algorithm like Hogwild!, the gain growth is the difference between the cost. For example, when we use $m$ workers, each worker trains $n_{local_m}$ iterations to reach the point of convergence. When we use $m+1$ workers, each worker trains $n_{local_{m+1}}$ iterations to reach the point of convergence. Thus, the gain growth is $n_{local_{m+1}} - n_{local_m}$.

Example 4: Using real-sim dataset, 8 equal performance workers and other stable algorithm setting on Hogwild! algorithm, server uses 6242 iterations to reach the point of convergence. In this case, the cost is the number of iterations for each worker: $6242/8 = 781$ iterations per worker. Using the real-sim dataset and 9 equal performance workers, server uses 6497 iterations to reach the point of convergence. Thus, the gain growth is $781 - 722 = 59$ iterations. As we can see from this example, although server have to train more iterations, yet the number of iterations per worker is decreasing.

For mini-batch SGD, DADA and ECD-PSGD, the gain growth is the value of goal function’s difference. For example, logloss decrease between using $m$ nodes and $m+1$ nodes at a fixed iteration.

Example 5: Using HIGGS dataset, 2 workers and other stable algorithm setting on mini batch SGD algorithm, at 50 server iteration, the logloss for this model is 4.7525. Using HIGGS dataset, 3 workers and other stable algorithm setting on mini-batch SGD, at 50 server iteration, the logloss for this model is 4.5871. The gain growth is $\|4.7525 - 4.5871\| = 0.1654$

The Upper Bound of Algorithm Scalability Base on the gain and gain growth, the upper bound of algorithm scalability, $m_{\text{max}}$, is to describe the following two situations:

1. Under perfect computer assumption, with the increasing of the number of nodes at the range $[m_{\text{max}}, \inf]$, the gain growth is positive but close to zero. In this case, the gain growth would not cover the parallel cost in a real computer. This situation suits ASGD algorithm, like mini-batch SGD, DADM and ECD-PSGD.

Example 6: Using HIGGS dataset and other stable algorithm setting on mini-batch SGD, the gain growth at 150 iteration is the 0.0011, 0.0006, 0.0003, 0.0002, 0.00018 match to the algorithm setting whose number of worker is 14, 15, 16, 17, 18, 19. As we can see from this case, the gain growth is decreasing (to zero). Thus, when the growth cannot cover the parallel cost, the system meets its scalability upper bound.

2. Under perfect computer assumption, with the increasing of the number of nodes at the range $[m_{\text{max}}, \inf]$, the gain is decreasing, or cost is increasing drastically. This situation suits the algorithms like Hogwild!.

Example 7: Using HIGGS dataset and other stable algorithm setting on Hogwild! algorithm, the gain growth is 14, 4, -7, -39, -72 match to the algorithm setting whose number of worker is 3, 4, 5, 6, 7. As we can see from this case, the gain growth
is decreasing. Thus, when gain growth is negative, the system meets its scalability upper bound.

**Measure index** In our experiments, we also use following index to measure the \( m_{\text{max}} \).

In this paper experiments, we use the following index to measure the ability of scalability: When the algorithm reaches the point of convergence, in Hogwild! experiment, we will use the iteration number of each worker as our index to measure the effect of parallel. When the algorithm reaches the point of convergence, in mini-batch SGD, DADM and ECD-PSGD, we use the iteration number for the server as the index to measure the effect of parallel.

When we use \( m \) worker, this index is \( \text{index}_1 \). When we use \( 2m \) worker, this index is \( \text{index}_2 \). When \( \text{index}_1 < \text{index}_2 \), we can know that \( m < m_{\text{max}} < 2m \).

**V. Convergence Analysis**

In this section, we will show the analysis conclusion for the above algorithms.

**A. Analysis**

In this section, we will show that the following conclusion in the theory aspect.

1. Different datasets suit different parallel machine learning training methods. Feature variance, sparsity and sample diversity can roughly classify datasets into different suitable algorithms. Besides high diversity datasets suit to DADM, we also show the following figure.

![Figure 1. Different datasets suit different parallel training methods](image)

2. The algorithm scalability performance for the same algorithm can be various depending on \( C_{\text{sim}} \).

3. The character of datasets decides the upper bound of algorithm scalability.

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

**B. Hogwild!**

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 \( \tau \) is the number of workers, i.e. \( m \leq \tau_{\text{max}} \). 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. 3

**Theorem 2:** Suppose in algorithm that the lag, i.e. \( \tau \), which is between when a gradient is computed and when it is used, is always less than or equal to \( \tau_{\text{max}} \), and \( \gamma \) is under certain condition. for some \( \epsilon > 0 \). When \( t \) is an integer satisfying

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

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

In theorem 2 \( \rho \) is the probability that any two \( G_{\xi}(x_i) \) and \( G_{\xi}(x_i) \) have the same nonzero value at the same feature; \( \Omega \) is the max number of nonzero feature in \( G_{\xi}(x) \); \( \delta \) 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 \( t/m = (1/m + 6\rho + 6m\delta^{1/2})\Omega h(\epsilon) \) 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 \( 1/(m + 1) + 6(1/m + 1)\Omega \delta^{1/2} < 1/m + 6m\delta^{1/2} \), 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 \( \Omega \) and \( \delta \). From the definition of \( \Omega \), \( \delta \) and \( \rho \), we can gain conclude that \( \Omega \delta \) 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 \( G_{\xi}(x) \) is equal to the sparsity of \( \xi \).

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 \( C_{\text{sim}} \) on \( \tau_{\text{max}} \)** In this algorithm, the sample sequence we discuss is the sequence the server receives from
workers. For example, server receives gradient sequence is \( G_{\xi_1}(x_1), G_{\xi_2}(x_2) \ldots G_{\xi_i}(x_i) \). Then, the sample sequence which we focus on is \( \xi_1, \xi_2, \ldots, \xi_i \) and the range is \( \tau_{\max} \).

The influence of \( C_{\text{sim}}\tau_{\max} \) is buried in the proof of theorem 2. The conclusion is that \( C_{\text{sim}}\tau_{\max} \) 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 3.

**The upper bound of scalability** From theorem 2 we draw the scalability upper bound which is decided by the character of dataset. To make time faster, at least each worker should train less sample compared with one worker, i.e. \( 1/m + 6m\Omega^3/2 < 1/1 + 6 \cdot 1 + \Omega^3/2 \). However, the function \( \text{constant}_1 \cdot \text{constant}_2 / x \) (\( \text{constant}_1, \text{constant}_2 > 0 \)) is increasing function when \( x \) is large enough. Thus the maximum of \( m \), which satisfies \( 1/m + 6m\Omega^3/2 < 1/1 + 6 \cdot 1 + \Omega^3/2 \) is the maximum number of worker we can use in Hogwild! The upper bound of Hogwild! scalability suits second situation in “The Upper Bound of Algorithm” section.

### C. Mini-batch SGD

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 goal function Eq. 1 is running on algorithm 2 then we have

\[
\mathbb{E}_{x \in D, t} f(x_t) - f(x^*) \leq \left( h_2(F(\cdot)) \left( d(\mu_{D^1}, \mu_{D^2}) + \sigma_{D^1} + W_2(D^0, D^*) (1 - \gamma \lambda)^t \right) \right)^2 + h_3(F(\cdot))
\]

where \( D^1 \) is the distribution of \( x^1 \), \( D^* \) is the distribution of \( x^* \), \( \sigma_0 \) is the standard deviation of distribution \( D \), \( \mu_0 \) is the mean of distribution \( D \), \( W_2(D_1, D_2) \) is the Wasserstein metrics between \( D_1 \) and \( D_2 \).

**Sparsity and Feature variance** When dataset and machine are chosen, \( D^0 \) and \( D^* \) would be determined. For most of the cases, the \( x^* \) is a fixed number. The value of \( W_2(D^0, D^*) \) is determined by the character \( D^0 \). Based on the definition of Wasserstein metrics, we can know that \( W_2(D^0, D^*) \) is positive correlate 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 \( C_{\text{sim}} \)** 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 \( C_{\text{sim}} \). For example, in algorithm 2 batch size is 3 and the sequence of server received \( G_{\xi}(x) \) is \( \{G_{\xi_1}(x_1), G_{\xi_2}(x_2), G_{\xi_3}(x_2), G_{\xi_2}(x_2), G_{\xi_3}(x_2), \ldots \} \), \( \{G_{\xi_{1-2}}(x_1), G_{\xi_{2-1}}(x_2), G_{\xi_{3-1}}(x_1), \} \), where the sample or gradient in \( \{\cdots, \cdots\} \) is in on batch. Then, the \( C_{\text{sim}} \) for mini-batch SGD algorithm is the \( C_{\text{sim}} \) for \( \xi_1, \xi_{1+1}, \xi_2 \) and \( \xi_1, \xi_{1+1}, \xi_2 \) can build a sequence whose \( C_{\text{sim}} \) is the maximum in all batches.

\( C_{\text{sim}} \) 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 \( C_{\text{sim}} \) 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 \( \frac{1}{\text{batch size}} \), 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 “The Upper Bound of Algorithm” section.

### D. ECD-PSGD

To present the convergence analysis of algorithm 4, we have to rewrite the goal function in to following form

\[
f(x) = \frac{1}{n} \sum_{i=1}^{n} F(x; \xi_i) = \frac{1}{m} \sum_{j=1}^{m} \frac{1}{n_{\text{local}}} \sum_{i=0}^{n_{\text{local}}} F(x; \xi_{i,j}) \tag{4}
\]

And we also define following notes:

\[
f_i(x) := \frac{1}{n_{\text{local}}} \sum_{i=0}^{n_{\text{local}}} F(x; \xi_{i,j}) \tag{5}
\]

\[
\sigma^2 \geq \frac{1}{n_{\text{local}}} \sum_{j=1}^{m} \| \nabla F(x; \xi_j) - \nabla f_i(x) \|^2, \forall x
\]

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

\[
\mathbb{E} \left( C(z^{(i)}_t) - z^{(i)}_t \right) = 0, \forall x, \forall t, \forall i
\]

\[
\hat{\sigma}^2 \geq 2 \mathbb{E} \left\| C(z^{(i)}_{t+1}) - z^{(i)}_t \right\|^2, \forall x, \forall t, \forall i
\]
For algorithm 4, Hanlin T et al.\cite{7} gives following convergence theorem.

**Theorem 4:** In algorithm 4 choosing an appropriate $\gamma$, it admits

$$
\frac{1}{T} \sum_{t=1}^{T} \mathbb{E} \|\nabla f(\bar{x})\| \leq \frac{\sigma}{\sqrt{mT}} + \frac{\sigma^2 \log T}{m^2} + \frac{\zeta^2/3}{\gamma^2} \log T + h_4(\bar{\sigma}, \zeta).
$$

(6)

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, $x = C(x), t \rightarrow \infty$, ECD-PSGD degenerates into mini-batch SGD. Thus, ECD-PSGD inherits the character 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 $\bar{\sigma}$, which means the ECD-PSGD is apt to accelerate the dataset, which would lose their a lot accurate during compress process.

**The influence of $C_{\text{sim}}$** 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 characters 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 "The Upper Bound of Algorithm" section.

**E. DADM**

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. 5 at each iteration and then broadcast its information to other workers to solve global problem $f(x)$ in Eq. 4 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 8.

**Theorem 5:** $f(\cdot), \xi_i$ and $\Delta x_{\text{local}}$ satisfy some requirements. When $t$ satisfies following condition, the expected duality gap of goal function and its dual form is smaller than $\epsilon$

$$
t \geq \frac{1}{\text{local}_\text{batch}_\text{size} \times m} (\log (h_5(\xi_i, \gamma, \lambda) + \epsilon) + h_6(x_0, \epsilon)).
$$

**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. 5 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 $C_{\text{sim}}$** The influence of similarity is hard to be shown in theory analysis. However, from algorithm 5 description step 2 in SERVER part, we can observe that using the definition of $C_{\text{sim}}$ in mini-batch SGD, when $C_{\text{sim}}$ is small, $v_i^{\text{local}}$s from a different worker would be almost the same, which would decrease the influence of parallel. Above fact suggest that when $C_{\text{sim}}$ is small, the parallel effect is poor.

**The upper bound of scalability** Again, the upper bound of scalability for DADM shares the same characters 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 "The Upper Bound of Algorithm" section.

**F. Theory Conclusion**

Based on our analysis, we can draw the following conclusion clearly:

1. Different datasets suit different parallel optimization algorithms.
2. The similarity is poison for the parallel optimization algorithm. Thus, before training a machine learning model, rearrange dataset is an excellent choice.
3. No matter which parallel optimization algorithm is, there always exists an upper bound of scalability. Moreover, the upper bound of scalability is determined by the dataset characters. Those characters can be described by dataset sparsity, feature variance and sample diversity.

**VI. Experiment**

In experiments, we will show the convergence curve on the figure whose X-axis is the number of iteration and Y-axis is the log loss. We do not want to compare which algorithm can find the $x^*$ better. We want to compare the scalability of the algorithm. Thus, we do not fine-grained control the algorithm parameters.

In our figure, the gap can indicate the effect of parallel technology. The upper bound of algorithm scalability has two
situations. So, different algorithms have different index to determine the scalability effect of the parallel algorithm:

For ASGD, i.e. Hogwild!, the effect is better when the gap is smaller: The number of iteration to reach a fixed $\varepsilon$ is stable when increasing the number of workers. Then the number of iteration in each node will decrease.

For ECD-PSGD and mini-batch SGD, the effect is better when the gap is large: At the fixed iteration, the log loss from a particular algorithm worker setting is smaller.

A. Experiments Setting

1) Dataset: We choose a sparse dataset with small feature variance as experiments dataset: real-sim dataset and a dense dataset with large feature variance: HIGGS dataset. The detail information about the above datasets is shown in table I. Their suitable algorithms are shown in figure 2.

In all cases, the dataset is randomly split into two parts: a training set containing 70% of the dataset samples and a valid set containing 30% of the dataset samples.

| dataset       | #features | size   | feature range | density |
|---------------|-----------|--------|---------------|---------|
| real-sim      | 20,958    | 72,309 | (0,1)         | < 3%    |
| HIGGS         | 28        | 11,000,000 | [-4,3]     | 100%   |
| Simulated Data | 20,958    | -      | 0/1           | 70%    |
| Simulated Data: Small $C_{sim}$ dense dataset | 28/1000 | - | [-4,3] | 100% |
| Simulated Data: Large $C_{sim}$ dense dataset | 28/1000 | - | [-4,3] | 100% |
| Simulated Data: Small $C_{sim}$ sparse dataset | 20,958 | - | [0,1] | < 3% |
| Simulated Data: Large $C_{sim}$ sparse dataset | 20,958 | - | [0,1] | < 3% |

Table I
Dataset Information

![Figure 2. The best performance dataset for different algorithm](image)

To match our theory, we also build three simulated dataset: (1) Small $C_{sim}$ dataset and (2) Large $C_{sim}$ dataset (3) Normal dataset for upper bound experiments. The samples in those dataset is generated randomly and the label is generated by the function $\text{label}_i = \text{sign}(\xi_i \cdot \text{ruler})$ where $\text{ruler}$ is the vector $(-1, 2, -3, 4, ..., (-1)^{\text{sample-size} \cdot \text{sample-size}})$.

Small $C_{sim}$ dataset and large $C_{sim}$ dataset Small $C_{sim}$ dataset and large $C_{sim}$ dataset are used to match the $C_{sim}$ related theory. All of information is shown in table I. In $C_{sim}$ experiments, the size of the test dataset is 20% of the number of training data. And the data in test data only share the same feature range and density character with training data.

In small $C_{sim}$ and dense dataset, the sample offered by $t$-th iteration is modified by the sample at $t-1$-th iteration: we randomly choose 10% features and randomly change those feature’s value.

In large $C_{sim}$ and dense dataset, the sample offered by $t$-th iteration is modified by the sample at $t-1$-th iteration: we randomly choose 90% features and randomly change those feature’s value.

In small $C_{sim}$ and sparse dataset, the sample offered by $t$-th iteration is modified by the sample at $t-1$-th iteration: we randomly choose 90% features and randomly change those feature’s value. To make sample sparse, we also randomly pick some feature and set them as zero, and the sparsity is equal to the sparsity the sample at 1-th iteration.

In large $C_{sim}$ and sparse dataset, the sample offered by $t$-th iteration is modified by the sample at $t-1$-th iteration: we randomly choose 90% features and randomly change those feature’s value. To make sample sparse, we also randomly pick some feature and set them as zero, and the sparsity is equal to the sparsity the sample at 1-th iteration.

Simulated Dataset for upper bound dataset For the upper bound of Hogwild! scalability on real-sim exceeds the number of cores of our computing environment. So we have to build a simulated dataset whose upper bound of scalability is easy to reach. In our simulated dataset, the density is 70%. Other information is shown in table I.

In scalability upper bound experiments, the size of the test dataset is 20% of the number of training data. And the data in test data only share the same feature range and density character with training data.

2) Hardware: We conducted our experiments on a server with 2 Xeon(R) CPU E5-2660 v2 @ 2.20 GHz, and 60G memory which contains twenty four cores together.

Because our experiment hardware is limited, we cannot conduct DADM experiments in our server: DADM requires that all samples load to memory at once, i.e. solve the subproblem minimum. Thus in the current version, we only present Hogwild!, mini-batch SGD and ECD-PSGD’s experimental results.

3) Problem: In our experiment, we will solve the problem of training L2 norm logistic regression model, because the log loss function suits all requirements which are asked by Hogwild!, mini-batch SGD and ECD-PSGD. The logistic loss function is shown in Eq. 7.

$$\argmin_x \frac{1}{n} \sum_{i=1}^{n} \Phi(\text{table}_i \cdot \xi_i \cdot x) + \frac{\lambda}{2} \|x\|^2$$

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

![Figure 2. The best performance dataset for different algorithm](image)
B. Feature variance and Sparsity Experiment

1) Algorithm Setting: In this experiment, we run HIGGS and real-sim on different algorithms to make the comparison. In Hogwild!, \( \tau \) is set to the same number, which is equal to the number of workers. Learning rate is 0.1. In the mini-batch SGD and ECD-PSGD, learning rates are 0.1. In Hogwild! experiments on HIGGS dataset, to gain a stable curve, we have to set the mini-batch as 16. In ECD-PSGD experiment, we connect all workers into a ring and we do not compress the data.

2) Experimental Results: The experimental results are shown in Figure 3, 4, 5.

3) Experiment analysis: In our feature variance and sparsity experiment, our experiment results well match to theory analysis: our experiment results match the figure 1. In mini-batch SGD and ECD-PSGD, the parallel effect is markable for large variance dataset (HIGGS). Large batch setting mini-batch SGD convergence faster, while for the sparse dataset (real-sim), the parallel technology does not exert any influence on convergence speed. For ECD-PSGD, parallel technology even brings a negative impact. (2) For ASGD algorithm, i.e. Hogwild!, with the increasing number of workers, the influence on convergence speed is minor on the sparse dataset. The iteration number on each node is decreased linearly. However, for feature variance dataset (HIGGS), the convergence speed is drastically decreasing, which means the iteration number on each worker is not reduced obviously. In some cases, the iteration number on each worker is increasing with the number of workers’ increasing.

C. C_sim Experiment

1) Algorithm Setting: The algorithm setting in this section is the same with feature variance section. The above sections show that different dataset suit different algorithm, we only present: 1. sparse dataset for Hogwild! 2. feature variance dataset for mini-batch SGD and ECD-PSGD. In Hogwild! experiment, the first sample is sampled from the real-sim dataset. In the mini-batch SGD experiment, the first sample is sampled from HIGGS dataset. In ECD-PSGD experiment, we use our own first sample to make the gap between different lines large which size is 1000.

2) Experimental Results: The experimental results are shown in Figure 6, 7, 8.

3) Experiment analysis: In our C_sim experiment, our experiment results well match to theory analysis: large C_sim
value leads to better scalability. In mini-batch SGD and ECD-PSGD, when $C_{\text{sim}}$ is large, at the same iteration, the more gain growth we can get: the gap between the different line is large. For ASGD algorithm, i.e. Hogwild!, when $C_{\text{sim}}$ is large, the more gain growth we can get: the gap between the different line is small, which means each worker trains fewer iterations.

D. Scalability Upper Bound Experiment

1) Algorithm Setting: The algorithm setting in this section is the same with feature variance section. Above sections show that different dataset suit different algorithm, we only present: 1. sparse dataset for Hogwild! 2. feature variance dataset for mini-batch SGD and ECD-PSGD.

Our experiment environment cannot reach the upper bound of scalability of the real-sim dataset: our experiments environment only supports twenty four thread (worker) in all. Thus, in Hogwild! experiment, we have to use simulated dataset. In the mini-batch SGD and ECD-PSGD experiment, we use HIGGS dataset.

We use the measure index as our index in this experiment. The definition of measure index is presented in “Measure Index” subsection in “Background Section”, ”The Upper Bound of Algorithm Scalability SubSection”.

2) Experimental Results: Base on the measure index in above subsection, we get following table II

| Algorithm | 1worker | 2workers | 4workers | 8workers | 16workers |
|-----------|---------|----------|----------|----------|-----------|
| Hogwild!  | 424     | 376      | 321      | 356      | 412       |
| mini-batch| > 200   | 91       | 87       | 71       | 69        |
| ECD-PSGD  | 1700    | 1654     | 1621     | 1623     | 1648      |

3) Experiment analysis: In table II we show that the different algorithms have their upper bound scalability, which is marked by red in table II even using their best performance dataset. Based on our analysis in “The Upper Bound of Scalability” Section, the growth gain for Hogwild! is negative. For ECD-PSGD and mini-batch SGD, the growth is close to zero. Thus, in the range which we marked, the algorithms meet their scalability upper bounds.

VII. CONCLUSION AND DISCUSS

A. Discuss

1. Some machine learning models, like CNN or DNN, do not obey the convex, Lipschitz or continuity requirements. Thus, the scalability of the algorithm on those model is needed to be analyzed.

2. For most cases, according to the experiments of papers [5, 7, 23, 8] and our investigation, the upper bound of the algorithm’s scalability is about 100 nodes. However, for a large-scale parallel computing environment, they contain more than 1,000,000 nodes. Above facts shows that current parallel machine learning training algorithms are not ready to run on those platforms.

3. Pre-process is not only crucial for machine learning accurate but scalability as well.

4. one scalability performance of an algorithm on a specific dataset cannot be pushed into other datasets.

B. Conclusion

Based on our analysis and experiments, we can draw the following conclusion clearly:

1. Different datasets suit different parallel optimization algorithm.

2. The similarity is poison for the parallel optimization algorithm. Thus, before training a machine learning model, rearrange dataset is an ideal choice.

3. No matter which parallel optimization algorithm is, there always exists an upper bound of scalability. Moreover, the dataset characters play critical roles in the upper bound of scalability. Those characters at least include dataset sparsity, feature variance and sample diversity.

VIII. ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant No. 61432018, Grant No. 61502450, Grant No. 61521092, and Grant No. 61272136, and by the National Major Research High Performance Computing Program of China under Grant No. 2016YFB0200800. State Key Laboratory of Computer Architecture Foundation under Grant No. CARCH3504

This work is finished during the Cheng Daning and Zhang Hanping’s internship in Wisdom Uranium technology Co.Ltd. Corresponding author is Shigang Li

APPENDIX

Theorem 1 The minimum of maximum of $\tau$ is the number of workers, i.e. $m \leq \tau_{\text{max}}$

proof In a $M$ worker cluster, for slowest worker, at $t$th iteration, this slowest worker submit its gradient to server. 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.
Theorem 3 When goal function Eq. \[1\] is running on algorithm\[2\] then we have

$$E_{x_i \in D_i} f(x_i) - f(x^*) \leq \left( (h_2(F(\cdot))) \left( d(\mu_{D^*}, \mu_{D^*}) + \frac{\sigma_{D^*} + W_2(D^0, D^*)(1 - \gamma \lambda)^t}{(\text{batch size})^{t/2}} \right) \right)^2 + h_3(F(\cdot)) \tag{9}$$

where $D^t$ is the distribution of $x^t$, $D^*$ is the distribution of $x^*$, $\text{sigma}_D$ is the standard deviation of distribution $D$, $\mu_D$ is the mean of distribution $D$. $W_2(D_1, D_2)$ is the wasserstein distance between $D_1$ and $D_2$.

**Proof 1:** Based on the work by M.Zinkevich et al.\[25\], we treat $x_i$ as random variable firstly and its distribution is $D_i$.

We have following theorem (Theorem 11 in M.Zinkevich\[25\])

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 $\sigma_D$ is bounded, then for any $v$

$$E_{x \in D}[f(x)] - \min f(x) \leq \sqrt{(W_2(v, D) \|\nabla f\|_L \min f(x)) + \frac{\|\nabla f\|_L (W_2(v, D)\|\nabla f\|_L)}{2} + (f(v) - \min f(x))} \tag{10}$$

When $v = x^*$. $W_2(\mu_D^*, D)$ is the relative standard deviation of $x_i$ with respect to $\mu_D$, i.e. $\sigma_D^{\mu_D^*}$.

Based on Theorem 32 in M.Zinkevich et al.\[25\], we know that

$$\sigma_D^{\mu_D^*} \leq \sigma_D + d(\mu_D^*, \mu_D) \tag{11}$$

$$\sigma_D^2 \leq \sigma_D + W(D^*, D^0)(1 - \gamma \lambda)^t \tag{12}$$

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 the definition, before average operation, $x_i$ is independent and identically distributed random variable. In each iteration, $\sigma_{D^t}$ is shrunk by $1/(\text{batch size})$. Combining above equations, we can get theorem.

**Lemma** $C_{\text{Sim}_{\tau_{\text{max}}}}$ is positively correlated to the scalability in Hogwild!

**Proof 2:** In the Hogwild! proof, the $\tau$ is created in following equation, figure\[3\] In following equation, $\delta, \Omega, \rho$ are create by the sum of multiplication of gradient $G_\xi$, or model difference ($x_i - x_{k(i)}$, which can be desrcibed as $G_\xi$). The sum range is $\xi_i$ to $\xi_i - \tau$. Above facts show that the original definition $\delta, \Omega, \rho$ is large: it is unnecessary to calculate those parameters in whole dataset. Just it is better define those parameter server in sample sequence neighborhood $\tau_{\text{max}}$ samples sub-dataset: If we define $\delta, \Omega, \rho$ as $\delta, \Omega_{\text{local}}, \rho_{\text{local}}$, which is calculated in sample sequence neighborhood $\tau_{\text{max}}$ samples sub-dataset and replace $\delta, \Omega, \rho$ in Hogwild! proof, the whole proof of Hogwild! is still sound. So, we find a tighter upper bound of Hogwild! algorithm.

![A8 in Hogwild! proof](a)

![part of Hogwild! proof](b)

![A6 in Hogwild! proof](c)

Figure 9. The proof segment of Hogwild!

As we can see from the definition, when $C_{\text{Sim}_{\tau_{\text{max}}}}$ is small, $\theta_{\text{local}}, \Omega_{\text{local}}, \rho_{\text{local}}$ is also small, which would increase the scalability ability.

**References**

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

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

[3] J. R. M. M. N. L. 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.

[4] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” 2015.

[5] 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.

[6] J. C. Duchi, *Introductory Lectures on Statistical Convex Optimization*. Park City Mathematics Institute, Graduate Summer School Lectures, 2016.

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

[8] 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.

[9] Y. M. Ermoliev, “On the stochastic quasi-gradient method and stochastic quasi-fejer sequences,” 1969.

[10] ———, “On the stochastic quasi-gradient method and stochastic quasi-fejer sequences,” 1969.

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

[12] “On the stochastic quasi-gradient method and stochastic quasi-fejer sequences.”

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

[14] TensorFlow, “Tensorflow: Google open sources their machine learning tool.”
