Neurocomputing 550 (2023) 126456

Contents lists available at ScienceDirect

Neurocomputing

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

Communication-efficient ADMM-based distributed algorithms for sparse training

Guozheng Wang, Yongmei Lei *, Yongwen Qiu, Lingfei Lou, Yixin Li

School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China

ARTICLE INFO

Article history:
Received 14 August 2022
Revised 27 May 2023
Accepted 11 June 2023
Available online 21 June 2023
Communicated by Zidong Wang

Keywords:
ADMM
Grouped Sparse AllReduce
Two-dimensional torus topology
Synchronization algorithm

ABSTRACT

In large-scale distributed machine learning (DML), the synchronization efficiency of the distributed algorithm becomes a critical factor that affects the training time of machine learning models as the computing scale increases. To address this challenge, we propose a novel algorithm called **G**rouped **S**parse **A**llReduce based on the **2D-T**orus topology (**2D-TGSA**), which enables constant transmission traffic that does not change with the number of workers. Our experimental results demonstrate that 2D-TGSA outperforms several benchmark algorithms in terms of synchronization efficiency. Moreover, we integrate the general form consistent ADMM with 2D-TGSA to develop a distributed algorithm (**2D-TGSA-ADMM**) that exhibits excellent scalability and can effectively handle large-scale distributed optimization problems. Furthermore, we enhance **2D-TGSA-ADMM** by adopting the resilient adap**t**ive **p**enalty parameter approach, resulting in a new algorithm called **2D-TGSA-TPADMM**. Our experiments on training the logistic regression model with ℓ_1-norm on the Tianhe-2 supercomputing platform demonstrate that our proposed algorithm can significantly reduce the synchronization time and training time compared to state-of-the-art methods.

1. Introduction

In recent years, the scale of data collection has reached unprecedented levels due to the rapid development of technology. To process these vast amounts of data and extract valuable knowledge from them, machine learning (ML) algorithms play a critical role [1]. Furthermore, the application of ML in emerging fields such as medical care [2], scientific simulations [3], and robotics [4,5] will generate more complex ML models and a wider range of data examples. These trends require increased computing power and storage capacity to train ML models effectively. As a result, large-scale parallel training on high-performance computing (HPC) systems are becoming more common to reduce the training time for larger models and datasets [6–8].

In modern ML, distributed optimization plays a crucial role, as it involves multiple computing nodes processing a vast amount of data and model parameters. The Alternating Direction Method of Multipliers (ADMM) [9] is an efficient distributed optimization method that can solve large-scale optimization problems. It is widely used in supervised machine learning problems such as regression and classification [10,11] due to its several distinct advantages. Firstly, ADMM combines with some high-precision optimization algorithms can achieve faster convergence speed in many applications [12]. Additionally, ADMM can decompose the global problem [13] into sub-problems and obtain the global solution by solving them in parallel. In a distributed implementation, sub-problems are deployed on multiple worker nodes, which greatly improves the efficiency of problem-solving. The communication structure in distributed machine learning mainly adopts the master–slave structure, such as the parameter server (PS), and the point-to-point structure, such as AllReduce. However, synchronizing model parameters is a communication-intensive process, and distributed training with a master–slave structure has poor scalability. The master node experiences excessive communication load and memory usage. On the other hand, distributed training in the peer-to-peer structure can balance the load of nodes. Here are several methods for enhancing the communication efficiency of distributed ADMM. AllReduce-based sparse communication [14,15] is a technique that reduces communication overhead. Compression techniques, such as CQ-GGADMM [16] and DQC-ADMM [17], have been proposed to reduce communication overhead through quantization and communication-censoring.

* Corresponding author.
E-mail addresses: gzh.wang@outlook.com (G. Wang), Lei@shu.edu.cn (Y. Lei), wen_2021@shu.edu.cn (Y. Qiu), loulingfei123@shu.edu.cn (L. Lou), liyixin0201@163.com (Y. Li).

https://doi.org/10.1016/j.neucom.2023.126456
0925-2312/

In distributed training, the training time of the distributed cluster system is mainly affected by the synchronization time between workers and the computation time of the workers. The communication network topology plays a key role in the convergence theory of multi-agent optimization methods, and an effective synchronization algorithm can accelerate distributed training. Additionally, many models are trained with sparse datasets, where the data has many fields, but only some fields are filled with values. Based on these considerations, this paper explores the design of synchronization algorithms aimed at improving the efficiency of communication in distributed training. Firstly, a grouped sparse AllReduce algorithm based on the 2D-Torus topology (2D-TGSA) is proposed to improve the synchronization efficiency of distributed algorithms. Secondly, 2D-TGSA is combined with ADMM to design a new distributed algorithm, namely 2D-TGSA-ADMM, which has good scalability. Thirdly, to improve the training speed of 2D-TGSA-ADMM, we improve the residual balancing method (RB) and propose a resilient adaptive penalty parameter method. This method is applied to the 2D-TGSA-ADMM algorithm, resulting in the proposed 2D-TGSA-TPADMM algorithm. The 2D-TGSA-TPADMM algorithm reduces computation time without sacrificing the convergence precision in distributed training. The main contributions of this paper can be summarized as follows:

(1) Firstly, in order to address the communication overhead problem of transmitting the high-dimensional sparse models, we propose a grouped sparse AllReduce algorithm, named 2D-TGSA. We compare 2D-TGSA with a benchmark synchronization algorithm and find that 2D-TGSA has better synchronization efficiency.
(2) Secondly, we combine the general form consensus ADMM with the 2D-TGSA and propose a communication-efficient distributed algorithm named 2D-TGSA-ADMM for solving large-scale high-dimensional sparse optimization problems.
(3) Thirdly, we propose the resilient adaptive penalty parameter method, which can adaptively adjust the penalty parameters of 2D-TGSA-ADMM based on changes in the dual residuals. Combining this method with Top$K$ sparse computation, we can reduce the solving time of sub-problems and accelerate the convergence of 2D-TGSA-ADMM. We name the improved algorithm 2D-TGSA-TPADMM.
(4) The two ADMM-based distributed algorithms proposed in this paper are designed to solve large-scale logistic regression problems with ℓ_1-norm. Experimental results show that the proposed 2D-TGSA-ADMM and 2D-TGSA-TPADMM algorithms outperform state-of-the-art methods in terms of updating time and synchronization costs.

The rest of this paper is organized as follows. Section 2 provides a review of related work. Section 3 describes the motivation behind this research. Section 4 explains the preliminary concepts and problem formulations. Section 5 presents the algorithm design process of 2D-TGSA-TPADMM. Section 6 presents the experimental setting and the evaluation results. Finally, Section 7 concludes the paper.

2. Related Work

2.1. Synchronization Architecture

Distributed training systems have been broadly studied recently to scale up ML for big data and large models. Data parallelism and model parallelism are two common distributed training methods. Recently, people have proposed more advanced methods [18,19] to find parallelization strategies that are much more efficient than simple data parallelism and model parallelism. However, these methods often lower statistical efficiency, resulting in poorer generalization ability.

The process of distributed training includes the model training phase and model synchronization phase. In the synchronization phase, each node uses a synchronization architecture [20] (such as PS and AllReduce) to aggregate its local model parameters. PS-based synchronization is a centralized synchronization method. There are two roles for each server, namely, PS and worker [21]. The workers generate gradients for parameters and *push* them to PS. Then the PS aggregates the gradients from workers and waits for workers to *pull* them back. The flexible synchronization mode enables PS to support efficient sparse communications, where each worker only accesses a small part of the model parameters in one iteration [22,23]. However, training large models usually suffers from communication bottleneck in the PS node, and it is difficult for PS to benefit from efficient AllReduce routines designed for HPC network hardware. Ring-based AllReduce is a decentralized algorithm, which has been studied in DML scenarios [24,25]. It works in a mode of *scatter* + *gather* and shows excellent performance in many DML training workloads. However, it degenerates into inefficient AllGather primitives for sparse communication.

2.2. Efficient Sparse Collectives

As the size of the training model increases, model parameter synchronization becomes the dominant component in distributed training. AllReduce can significantly stall the calculations of the following training epoch and is sensitive to stragglers and communication delays. Therefore, it can quickly become a bottleneck for large-scale distributed training. To address this problem, there are two main research directions.

Firstly, several synchronization algorithms have been proposed for AllReduce operation [26,27]. Baidu Research implemented a bandwidth-optimal ring AllReduce algorithm [26], which has been included in the popular deep learning framework [24]. However, ring AllReduce suffers from long latency and may have low resource utilization in large-scale clusters. Huang et al. [27] presented the MultiTree algorithm, which uses a customized network architecture to minimize network contention and can provide contention-free communication. It only conducts communication over not-yet-occupied links, skipping the others to the next step. Miao et al. [25] proposed a novel variant of AllReduce that provides high heterogeneous tolerance and performance by decomposing the synchronous AllReduce primitive into parallel-asynchronous partial-reduce operations. Although the relaxing idea works well for transient stragglers, distributed algorithms that use asynchronous communication mechanisms may cause a certain degree of degradation in the test accuracy of the training model.

The second research direction for addressing the communication bottleneck in distributed training is sparse communication based on AllReduce. Shi et al. [28] proposed a mechanism called global Top$K$, where instead of using the top K gradients from each worker, only the top K gradients among all workers are used. Shi et al. [29] provided convergence results for this method. However, this technique usually takes into account the fixed sparsity or compression ratio that needs to be configured as a hyperparameter. Li et al. [30] designed a general communication library, SparCML, which extends MPI to support additional features such as non-blocking (asynchronous) operations and low-precision data representations. SparCML uses the AllReduce synchronization scheme to design a fast random top-k algorithm, but this does not mean top-k selection. Some important gradient elements are omitted from its communication set, which leads to the convergence problem of the system. Chen et al. [31] proposed the Scalable Sparsified Gradient Compression (ScaleCom) method, which tailors AllReduce to sparse data. The work either determines the compression level in

advance or adjusts the level according to a heuristic method, which may lead to contradictory conclusions. Fei et al. proposed OmniReduce [32], which partitions data into blocks and only sends nonzero data blocks, greatly reducing the overhead of transmitting indices, especially when used with block-based compression methods. However, it forces the application to sparsify the data perblock, i.e., to send sparse blocks of dense data, which negatively impacts the convergence of distributed training.

Although the proposed sparse AllReduce reduces communication overhead through some manual parameter-adjusting mechanisms, it negatively impacts the convergence of distributed algorithms, especially in large-scale cluster environments. In this paper, we further reduce the mutual waiting time and communication overhead among workers in large-scale clusters, mainly by substantially reducing communication traffic while ensuring the convergence accuracy of distributed algorithms.

2.3. Communication-efficient Distributed ADMM Algorithm

To enhance the communication efficiency of distributed learning, previous research has explored various technologies under centralized and decentralized network architectures. Liu et al. [17] attempted to combine communication censoring with ADMM to solve the decentralized dynamic consensus optimization problem. During the training process, the censoring ADMM algorithm uses a threshold to limit the transmission of unimportant information, which can reduce communication volumes during the optimization process. However, the inexact algorithm lacks a compensation mechanism resulting in poor convergence. Wang et al. [15] presented an asynchronous lazy ADMM algorithm based on hierarchical sparse AllReduce communication mode, tailored for sparse data to effectively aggregate the filtered transmission parameters. To improve communication efficiency, the algorithm adopts stale synchronous parallel (SSP) bridging model and filters out unnecessary information by manually setting a threshold. Although the communication efficiency is improved, the precision of the algorithm is sacrificed, and the threshold value needs to be selected manually, resulting in poor scalability of the algorithm. Several works have studied ADMM variants used for federate learning (FL). Specifically, to reduce communication and calculation burden, Zhou et al. [33,34] studied an ADMM variant with a flexible communication mechanism for FL, where clients only communicate with the central server at fixed time points instead of at each iteration. FL usually prioritizes the efficiency of calculation and communication over the accuracy of solutions. However, FL assumes that the parameter server collects and distributes model parameters, which are not always available from faraway workers and are easily affected by a single point of failure.

In DML training, although certain sparse communication strategies that involve manually setting the threshold can exclude more unnecessary transmission information, they may lead to lower training accuracy. Therefore, balancing the communication costs and accuracy of the training model is therefore a significant challenge.

3. Motivation

3.1. Communication Overhead

In high-dimensional settings, sparsity is a crucial property that can be leveraged to accelerate training. Sparsity can be divided into two aspects: data sparsity and model sparsity. In terms of data sparsity, datasets typically have a large number of features, most of which have zero values. If the feature representation of the current sample is zero, the stochastic gradient with respect to the corresponding feature must also be zero [35]. On the other hand, regularization typically enforces model sparsity, where the coefficients of associated features are zero at the optimum. Model sparsity can be used to reduce problem dimensionality by identifying inactive features beforehand [36]. Discarding these features can save significant computation without any loss of accuracy. Therefore, exploiting sparsity to accelerate high-dimensional models training is a promising and much-needed approach.

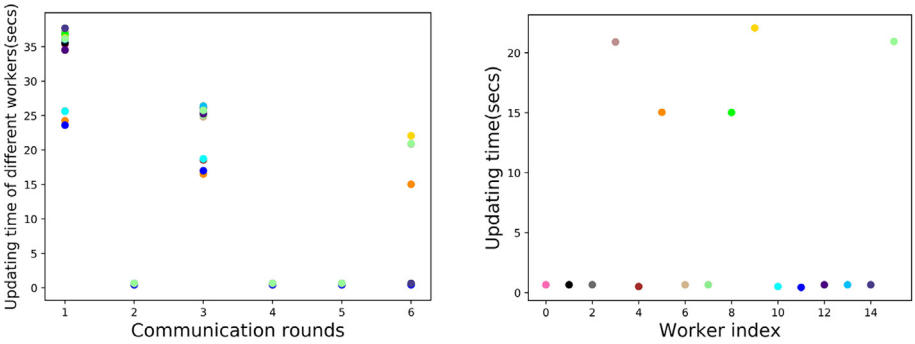
In the ADMM-based distributed algorithm, the aggregation of parameters for each iteration is performed among workers, which can consume heavy network bandwidth resources. The larger the model, the faster the bandwidth costs increase. AllReduce is an efficient and widely-supported collective synchronization operation that directly aggregates the parameters of workers. However, the communication primitives ignore that AllReduce is unsuitable for sparse transmission. In particular, it is easy to generate sparse models by training sparse datasets or using some regularization methods. Therefore, it is a waste of bandwidth to transmit many zeros in distributed training.

3.2. Training Speed Gap among Workers

Nowadays, the SSGD algorithm [37] is commonly used for training neural networks [38,18,19], as it is easy to implement and deploy. However, due to communication delays in each synchronization process and the mutual waiting between workers with different performances, a large amount of extra overhead is introduced [29]. Specifically, in each iteration, all workers use different mini-batch data to compute the gradients of the model in parallel and then average the gradients of each worker to update the global model later, which introduces a large communication overhead. In addition, after each iteration, due to the synchronization of local model parameters, the workers with strong performance have to halt, waiting for the workers with weak performance, which leads to high waiting costs. Sometimes, when the distributed SGD algorithm trains a larger model, the waiting time and synchronization time even exceed the updating time, which decreases the utilization rate of computing resources and limits the scalability of the algorithm [39].

The ADMM algorithm can decompose the primal problem into sub-problems for distributed computing. In ADMM-based distributed training, sub-problem solving using first-order algorithms usually suffers from high communication complexity. In contrast, second-order algorithms can have faster convergence speed, resulting in lower communication complexity [40]. The trust region Newton method (TRON) [41] has a quadratic convergence rate. However, in the synchronization process, the number of CG-Newton iterations can cause differences in training speed among different workers, leading to mutual waiting between workers. We experimentally verify this issue by training a logistic regression model using the ADMM algorithm on 16 workers. Fig. 1 provides an illustrative example.

The difference in workers' iteration time is mainly reflected in the sub-problem solving algorithm. In different colors, we show the time consumption by 16 workers in solving the sub-problems. As can be seen from Fig. 1(a), the iteration time of the sub-problem varies significantly as the ADMM algorithm performs the consistency operation. In the sixth consistency operation of the ADMM algorithm, the time consumption by each worker in calculating sub-problems is mapped in Fig. 1(b). It can be seen that the iteration time of all workers is between 0.43s and 24.03s. Some fast workers, such as `worker`5 (orange) and `worker`8 (green), can be calculated in about 15s, while some slow workers, such as `worker`3 (purple) and `worker`15 (light green), can be calculated in about 24s, which means that slow workers consume 37.5% more time than fast workers. In a large-scale cluster, fast workers need to wait, which will waste computing resources and slow down the convergence speed of distributed algorithms.



(a) TRON method used by each worker to train local model before calculating global model.

(b) Training time with 16 workers in the 6th communication round.

Fig. 1. Interval in training time with 16 workers.

4. Preliminary and Problem Formulation

4.1. Preliminary

Notations. $\mathbb{R}$ denotes the set of real numbers. The symbol "$\langle,\rangle$" means $\langle a,b\rangle=[a_1b_1,\ldots,a_nb_n]^T$, respectively, where $a=[a_1,\ldots,a_n]^T$ and $b=[b_1,\ldots,b_n]^T\in\mathbb{R}$ are column vectors. $\|\cdot\|$ denotes the Euclidean norm of vectors. $\mathscr{S}$ is the soft threshold and its definition is shown in Eq. (1).

$$\mathscr{S}_{\gamma/N\rho}(\phi)=\begin{cases}\phi-\gamma/N\rho, & \phi>\gamma/N\rho,\\ 0, & |\phi|\leqslant\gamma/N\rho,\\ \phi+\gamma/N\rho, & \phi<-\gamma/N\rho.\end{cases}\tag{1}$$

4.2. Problem Formulation

This paper mainly focuses on supervised machine learning problems, including classification and regression problems. The problem can be abstracted in the following form.

$$\min_{\boldsymbol{x}} l(\boldsymbol{x})+r(\boldsymbol{x}),\tag{2}$$

where $l(\boldsymbol{x})$ is the loss function and $r(\boldsymbol{x})$ is the regularization term. The regularization term typically includes ℓ_1-norm and ℓ_2-norm. ℓ_1-norm can generate a sparse weight matrix, which can be used for feature selection. ℓ_2-norm can prevent the model from overfitting.

General form consensus problem [9]. This paper focuses on the general form of the problem, in which the local variables $x_i\in\mathbb{R}^d, i=1,\cdots,N$ have a separable loss function $f_1(x_1)+\cdots+f_N(x_N)$ with respect to x_i. Each of these local variables consists of a selection of the components of the global variable $\boldsymbol{z}\in\mathbb{R}^d$, which we call $\tilde{z}_i$. The general form consensus problem with ℓ_1-norm, where $g(\tilde{z}_i)=\gamma\|\tilde{z}_i\|_1$, is as follows:

$$\begin{aligned}&\min_{x_i}\sum_{i=1}^{N}f_i(x_i)+g(\tilde{z}_i),\\ &\text{s.t.}\quad x_i-\tilde{z}_i=0, i=1,\cdots,N.\end{aligned}\tag{3}$$

By constructing a Lagrangian function as shown in Eq. (4), the constrained problem (3) is transformed into an unconstrained problem.

$$\mathscr{L}(\boldsymbol{x},\boldsymbol{z},\lambda)=\sum_{i=1}^{N}(f_i(x_i)+\langle\lambda_i,x_i-\tilde{z}_i\rangle+\frac{\rho}{2}\|x_i-\tilde{z}_i\|_2^2+\gamma\|\tilde{z}_i\|_1).\tag{4}$$

ADMM iterations [9]. The problem (4) is iteratively solved by the ADMM algorithm.

$$x_i^{k+1}:=\arg\min_{x_i}(l_i(x_i)+<\lambda_i^k,x_i>+\frac{\rho}{2}\|x_i-\tilde{z}_i^k)\|_2^2),\tag{5}$$

$$z_g^{k+1}:=\mathscr{S}_{\gamma/N\rho}(\frac{\sum_{\mathscr{G}(i,j)=g}((x_i^{k+1})_j+\frac{1}{\rho}(\lambda_i^k)_j)}{\sum_{\mathscr{G}(i,j)=g}1}),\tag{6}$$

$$\lambda_i^{k+1}:=\lambda_i^k+\rho(x_i^{k+1}-\tilde{z}_i^{k+1}),\tag{7}$$

where $\lambda=[\lambda_1,\cdots,\lambda_N]\in\mathbb{R}^d$ is the dual variable, and $\rho>0$ is a penalty parameter. The z-update step is decoupled across the components of $\boldsymbol{z}$ to reduce the communication frequency. To further reduce communication costs, x_i and λ_i are merged on the distributed memory, as shown in Eq. (8). Finally, z_g is obtained by averaging all entries of Eq. (8) that correspond to the global index g.

$$w_i^{k+1}=x_i^{k+1}+\frac{1}{\rho}\lambda_i^k,\tag{8}$$

ADMM residuals. Let x^* and z^* denote the optimal primal variables, and λ^* denote the optimal dual variable. The *primal residual*, as shown in Eq. (9), is defined as:

$$r^{k+1}=x^{k+1}-z^{k+1}.\tag{9}$$

Defining

$$s^{k+1}=\rho(z^{k+1}-z^k),\tag{10}$$

as the *dual residual*.

5. Communication Efficient Resilient Adaptive ADMM Algorithm

5.1. Sparse Synchronization Algorithm

Efficient communication topology is crucial to reducing the communication load of model parameter synchronization. Previous works propose ring-based topology and 2D-Torus topology to improve the efficiency of AllReduce operations. However, these synchronization algorithms have limitations. For instance, when using a synchronization algorithm such as Ring AllReduce, the faster worker needs to wait for the slower worker to finish the computation, which significantly reduces the efficiency of the distributed algorithm. Ring AllReduce and 2D-Torus AllReduce are susceptible to "slow nodes" and have low transmission utilization efficiency for large-scale clusters with thousands of CPUs. To address these issues, we propose a grouped sparse AllReduce algorithm based on 2D-Torus (2D-TGSA).

5.1.1. Grouped 2D-Torus AllReduce

The topology of 2D-TGSA is illustrated in Fig. 2. We use the updating situation of workers shown in Fig. 1(b) as an example.

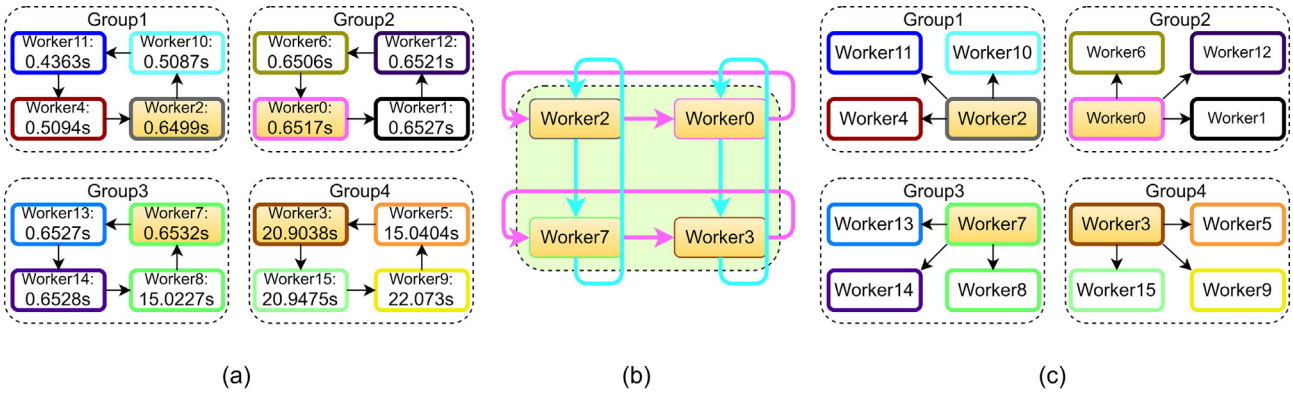


Fig. 2. (a) Workers are grouped based on their updating time, and the worker with the smallest rank in the group is selected as the leader. (b) The leader executes an improved 2D-Torus AllReduce. (c) The leaders broadcast the updated parameters to other workers in the group.

In this example, there are 16 workers in the cluster, and their training speeds vary greatly. We dynamically divide the workers into four groups, each containing four workers, with the worker having a smaller rank value in each group serving as the leader. The leaders are arranged in a 2D grid. 2D-TGSA consists of three steps. (1) The workers in the group perform a Ring AllReduce operation. (2) The leaders perform an improved 2D-Torus AllReduce operation. (3) The leaders then perform the broadcast operation within their group. We focus on the second step, as shown in Fig. 2(b).

In the 2D-Torus topology, reduction operation consists of three phases: ReduceScatter, Segmented Ring AllReduce (SRA), and AllGather, where the SRA is our improved method. Based on Ring AllReduce, data must be divided into blocks. Suppose there are N workers in the cluster $W_0, W_1, \ldots, W_{N-1}$. These workers are built on a Cartesian topology, with $\sqrt{N} = L$ workers in the horizontal and vertical directions. In the first phase, ReduceScatter is performed horizontally. Let W_i denote the i-th worker, W_i has E parameter elements $\{e_i^0, e_i^1, \ldots, e_i^{E-1}\}$, and the parameter elements in each worker are divided into L blocks, where $\{e_i^0, \ldots, e_i^{\lceil \frac{E}{L} \rceil}\} \in chk_0, \ldots, \{e_i^{(L-1)\lceil \frac{E}{L} \rceil}, \ldots, e_i^{E-1}\} \in chk_{L-1}$. The ReduceScatter operation requires $L-1$ iterations, where j is the number of iterations, $1 \leqslant j \leqslant L-1$. We use a generic operator $\oplus$ to denote the reduction operator, where

$$W_i[chk_{(i+\sqrt{L}-j)\%\sqrt{L}}] = W_i[chk_{(i+\sqrt{L}-j)\%\sqrt{L}}] \oplus W_{(i+\sqrt{L}-j)\%\sqrt{L}}[chk_{(i+\sqrt{L}-j)\%\sqrt{L}}]. \tag{11}$$

When the ReduceScatter operation executes for the j-th time, the chunk $chk_{(i+\sqrt{L}-j)\%\sqrt{L}}$ of neighbouring `worker` $(i+\sqrt{L}-j)\%\sqrt{L}$ are reduced to $chk_{(i+\sqrt{L}-j)\%\sqrt{L}}$ of `worker` i in the horizontal direction. In the second phase, the SRA operation is performed in the vertical direction, targeting a sub-block after the horizontal segmentation. This operation only needs to be executed on the $chk_{(i+\sqrt{L}-j)\%\sqrt{L}}$ block

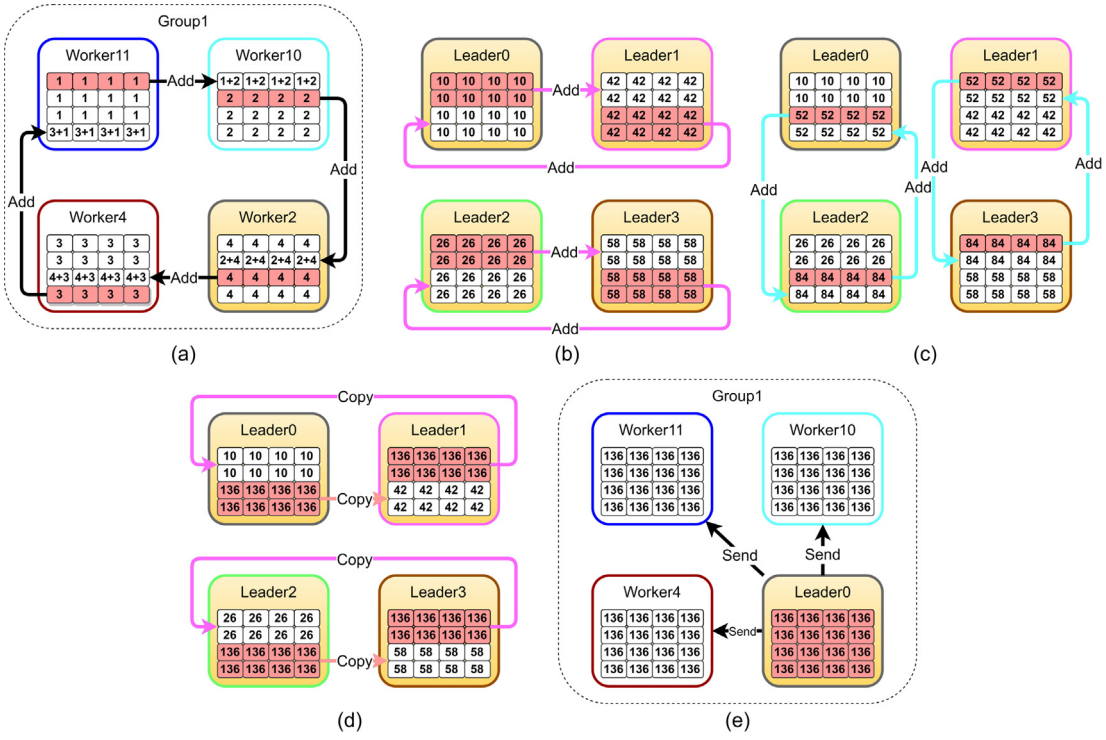


Fig. 3. (a) Workers in the group execute the Ring AllReduce operation. (b) The leaders execute the ReduceScatter operation horizontally. (c) The leaders execute the SRA operation vertically. (d) The leaders execute the AllGather operation horizontally. (e) The leader broadcasts the model parameters to the workers in the group.

of worker i, dividing the $chk_{(i+\sqrt{L}-j)\%\sqrt{L}}$ data block into $\sqrt{L}$ blocks, and performing a Ring AllReduce operation. Unlike the original 2D-Torus AllReduce, the newly proposed SRA only employs reduction operations by sub-blocks, significantly reducing the time spent on the reduction operation. In the third phase, all workers perform the AllGather operation in the horizontal direction.

We provide an example to illustrate the 2D-TGSA algorithm, as depicted in Fig. 3. In this instance, there are 16 workers, where $N = 16$, and each worker contains a vector with a model parameter size of 16, with $E = 16$. We divide these 16 workers into four groups, each consisting of four workers, as shown in Fig. 3(a), and select the minimum rank of each group as the leader. There are four leaders in total. These four leaders are arranged in a Cartesian topology, with four workers in the horizontal and vertical directions. Firstly, the ReduceScatter operation is executed horizontally, as shown in Fig. 3(b). Then, the SRA operation is executed vertically, as shown in Fig. 3(c). Later, the AllGather operation is executed horizontally, as shown in Fig. 3(d). Finally, as shown in Fig. 3(e), the leader broadcasts the updated model parameters to the workers in the group.

5.1.2. Data Representation and Messages Aggregation

Many models used in various applications operate on data with a large number of input fields, potentially numbering in the hundreds of thousands. However, only a few hundred of these features contain data, which are referred to as sparse data. Compared to dense data, sparse data typically has a much smaller number of non-zero entries. In DML, the sparsity of model parameters is mainly related to the datasets and objective function. Models that use sparse data present unique challenges, especially when high-dimensional models need to be transferred between workers. In networks with limited communication capabilities, the communication overhead can significantly affect the performance of DML tasks.

We encode sparse model parameters using < *key, value* > pairs. Suppose there are N workers in the cluster, and W_i represents the i-th worker. Each worker has d model parameters that require the AllReduce operation. Let E_i represent the set of non-zero elements on worker W_i. If the number of non-zero elements is much smaller than the dimension of model parameters (e.g., $E_i \ll d$), we can use the sparse transmission method.

For a dense set of model parameters, each element's position in the vector corresponds to the element's index, so it is necessary to store the value of each element. Assuming a d-dimensional vector where each element requires *vsize* bytes to store a value, storing the vector requires $vsize * d$ memory space. Sparse vectors can be stored using < *key, value* > pairs. For each non-zero element, you need to store *key* and *value*. Assuming it takes *isize* bytes to store a *key*, for a d-dimensional vector with sparsity σ, it only requires $(vsize + isize) * d * \sigma$ memory space.

5.1.3. 2D-TGSA Synchronization Algorithm

We divide the workers into groups based on the number of workers, aiming to have the number of leaders equal to the square root of the number of workers. Once the group size is determined, we can perform sparse Ring AllReduce operations within the group. After all the groups have completed the reduction, the worker with the smallest rank value in the group is selected as the leader. The leader constructs a Cartesian topology and performs sparse ReduceScatter operations in the horizontal direction, sparse SRA operations in the vertical direction, and finally, performs sparse AllGather operations in the horizontal direction. The specific algorithm flow is depicted in Algorithm 1.

Algorithm 1: 2D-TGSA on worker i

input : Worker numbers N
Group numbers L
Local parameters of each worker w_i
output: Global synchronization parameters $\boldsymbol{w}$

initialize j=1;
// Hierarchical mode
MPI_Comm_split();
// Leader group
MPI_Group_incl();
// Create subgroup and leadergroup
MPI_Comm_create_group();
// Create 2D torus topology
MPI_Cart_create();
// LEFT, RIGHT, UP, DOWN directions
MPI_Cart_shift();
do in parallel;
  // Intra-group RingAllReduce
  RingAllreduce();
  if worker i == leader **then**
    for $j \leftarrow 1$ **to** $\sqrt{L}$ **do** // perform sparse ReduceScatter horizontally
      ReduceScatter($w_i[(i+\sqrt{L}-j)\%\sqrt{L}]$,LEFT,RIGHT,$\sqrt{L}$);
    for $j \leftarrow 1$ **to** $\sqrt{L}$ **do** // perform sparse SRA vertically
      SegmentedRing($w_i[(i+\sqrt{L}-j)\%\sqrt{L}][(i+\sqrt{L}-j)\%\sqrt{L}]$,UP,DOWN,$\sqrt{L}$);
    for $j \leftarrow 1$ **to** $\sqrt{L}$ **do** // perform sparse AllGather horizontally
      AllGather($w_i[(i+\sqrt{L}-j)\%\sqrt{L}]$,LEFT,RIGHT,$\sqrt{L}$);
    MPI_Bcast();
return $\boldsymbol{w}$;

5.1.4. Evaluation

AllReduce is a synchronization operation used in DML that reduces target arrays in all workers to a single array and returns the resulting array to all workers. Synchronization among workers is one of the many challenges when training distributed machine learning models, and several typical algorithms, such as single ring and single tree, have been implemented to address this issue. We model several popular synchronization algorithms and analyze their efficiency. Table 2 shows the overall time consumption of various synchronization algorithms, where the bold formula represents the communication time consumption, and the non-bold formula represents the reduction time consumption. The symbols and parameter values used in our analysis are shown in Table 1.

The 2D-TGSA algorithm is based on the ReduceScatter and AllGather operations. Firstly, the data of each worker is divided into N blocks. The ReduceScatter operation is completed through $N-1$ steps, allowing each worker to obtain $\frac{1}{N}$ of the complete data block. The communication time consumption for each step is $\alpha + \frac{S\delta}{\beta N}$, and the reduction time is $\frac{S\delta C}{N}$. In the AllGather operation, every $\frac{1}{N}$ data block of all workers is completed in $N-1$ steps, with the commu-

nication time consumption per step also being $\alpha + \frac{S\delta}{\beta N}$. The overall synchronization time consumption is approximately $2(N-1)(\alpha + \frac{S\delta}{\beta N}) + (N-1)\frac{S\delta C}{N}$. The ReduceScatter and AllGather operations are advantageous because they ensure that the communication volume does not change with the increase in the number of nodes, resulting in a synchronization time that does not change sharply, as shown in Fig. 4. In addition, the 2D-TGSA algorithm has obvious advantages over the other two non-sparse algorithms because it only transfers non-zero parameters, making it particularly suitable for sparse training tasks.

5.2. Algorithm Development

In Section 5.1.3, we introduce the 2D-TGSA algorithm, which is designed to synchronize distributed training. We combine this algorithm with the general consensus ADMM algorithm to design a distributed algorithm, as shown in Algorithm 2. This algorithm is suitable for training sparse models on high-dimensional sparse datasets. It makes full use of distributed bandwidth and improves communication transmission efficiency.

Algorithm 2: 2D-TGSA-ADMM on worker i

input : The maximum iteration $\mathcal{K}$
Group numbers L
output: Global parameter z after iterations

initialize $w_i^0 = \mathbf{0}$, $x_i^0 = \mathbf{0}$, $\lambda_i^0 = \mathbf{0}$, $z^0 = \mathbf{0}$;
for $i \leftarrow 0$ **to** $\mathcal{K}$ **do**
  update x_i^{k+1} by (5);
  update w_i^{k+1} by (8);
  `// perform sparse synchronization algorithm`
  $\boldsymbol{w}^{k+1} \leftarrow$ 2D-TGSA(w_i^{k+1},L);
  update z_g^{k+1} by (6);
  update λ_i^{k+1} by (7);
  $k \leftarrow k+1$;
return $z^{\mathcal{K}}$;

Table 1
Notations.

Symbol	Description	Value
N	The total number of workers	128∼640
L	The number of groups	4
α	Delay between two communication workers	$0.7\mu s$
S	The total size of parameters on each worker	3231961
δ	Value data type (DOUBLE)	8B
ι	Key data type (INT)	4B
β	The bandwidth capacity of cluster	$7GB/s$
C	Computation time per byte of data	$1.6ns$
σ	Model parameter sparse rate	84%

5.2.1. Resilient Adaptive Penalty Parameter Method

The ADMM can decompose the original problem into sub-problems and solve them using high-precision algorithms. The TRON method [41] is a high-order algorithm that can achieve a quadratic convergence rate, but the updating time of the sub-problem is a relatively large part of the single iteration of the distributed ADMM algorithm. In Section 3, we analyze the computation speeds of sub-problems in distributed computing and find that they can lead to mutual waiting problems between workers in the synchronization process. To address this issue, we study adaptive ADMM in the distributed setting, where different workers use different local penalty parameters to accelerate convergence. He et al. [44] proposed an improved convergence rate for the ADMM algorithm by using an adaptive penalty term, which reduces the influence of the initial penalty value on the convergence performance. The idea is to consider the relative size of the primal residual and the dual residual of the ADMM algorithm, as shown in Eq. (12).

$$\rho^{k+1} = \begin{cases} \rho^k \cdot (1+\tau), & \text{if} \quad \|r^k\|_2 > \zeta\|s^k\|_2, \\ \rho^k \cdot (1+\tau)^{-1}, & \text{if} \quad \|s^k\|_2 > \zeta\|r^k\|_2, \\ \rho^k, & \text{otherwise}, \end{cases} \tag{12}$$

where k is the iteration index, $\zeta > 0, \tau > 0$ are parameters, r^k and s^k are the primal and dual residuals, respectively. The typical choice of parameters is $\zeta = 10$ and $\tau = 1$ for all iterations $\mathcal{K}$. However, this method has no constraints on the penalty parameter ρ and is not a general approach. As ρ increases, the algorithm may fail to converge. To address this issue, we propose a resilient adaptive penalty parameter method that improves the training speed of the algorithm while preventing the penalty parameter from deviating too far from its initial value, which can cause the ADMM algorithm to fail to converge. A simple scheme for achieving this goal is to update the penalty parameter as follows:

$$\rho^{k+1} = \begin{cases} \rho^k \cdot (1+\tau), & \text{if} \quad \|r^k\|_2 > \zeta\|s^k\|_2, \\ \rho^k \cdot (1+\tau)^{-1}, & \text{if} \quad \|s^k\|_2 > \zeta\|r^k\|_2, \\ \rho^0, & \text{otherwise}. \end{cases} \tag{13}$$

The parameters of the method follow the original paper [44].

5.2.2. Accelerated Residual Drop by TopK Sparsity

The idea of residual balancing (RB) [44] comes from increasing ρ^k to strengthen the penalty term, yielding smaller primal residuals but larger dual ones. On the contrary, decreasing ρ^k leads to larger primal and smaller dual residuals. As both residuals must be negligible at convergence, it makes sense to "balance" them [45]. The primal residual is the same as the dual ascent method's gradient. To be able to make the primal residuals fall fast, we use the Top$K$ algorithm to select the gradient information, which is essential for the dual variables, as shown in Eq. (14),

$$C_\kappa(G[j]) = \begin{cases} G[j], & \text{if} \quad j \in I_\kappa(G), \\ 0, & \text{otherwise}, \end{cases} \tag{14}$$

Table 2
Global synchronization time of various algorithms.

Synchronization algorithm	Overall time
WRHT [42]	$2\log_{\frac{N}{L}} N(\alpha + \frac{S\delta}{\beta} + S\delta C)$
Moshpit AllReduce [43]	$\log_2 \frac{N}{L}(\alpha + \frac{S\delta}{\beta} + S\delta C) + \log_2 L(\alpha + \frac{S\delta}{\beta} + S\delta C)$
HSA [15]	$\boldsymbol{2}(\alpha + \frac{\boldsymbol{S}(\delta+\iota)\sigma}{\beta}) + \boldsymbol{2}(\boldsymbol{L}-\boldsymbol{1})(\alpha + \frac{\boldsymbol{S}(\delta+\iota)\sigma}{\beta \boldsymbol{L}}) + \frac{N}{L}S(\delta+\iota)\sigma C + (L-1)\frac{S(\delta+\iota)\sigma C}{L}$
2D-TGSA	$\boldsymbol{2}(\frac{\boldsymbol{N}}{\boldsymbol{L}}-\boldsymbol{1})(\alpha + \frac{\boldsymbol{S}(\delta+\iota)\sigma \boldsymbol{L}}{\beta \boldsymbol{N}}) + (\alpha + \frac{\boldsymbol{S}(\delta+\iota)\sigma}{\beta}) + \boldsymbol{2}(\sqrt{\boldsymbol{L}}-\boldsymbol{1})(\alpha + \frac{\boldsymbol{S}(\delta+\iota)\sigma}{\beta\sqrt{\boldsymbol{L}}}) + \boldsymbol{2}(\sqrt{\boldsymbol{L}}-\boldsymbol{1})(\alpha + \frac{\boldsymbol{S}(\delta+\iota)\sigma}{\beta \boldsymbol{L}}) + (\sqrt{L}-1)\frac{S(\delta+\iota)\sigma C}{\sqrt{L}} + (\sqrt{L}-1)\frac{S(\delta+\iota)\sigma C}{L}$

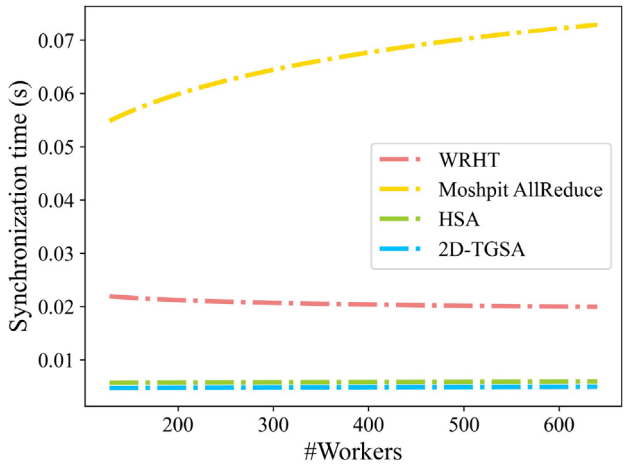


Fig. 4. Synchronization time of various algorithms as the number of workers increases.

where $C_\kappa(\cdot)$ is a compression operator, $I_\kappa(G)$ is the index set for the K components of G with largest magnitude [46]. In this case, we only calculate the gradient magnitude and the sparsity pattern of the gradient. Since it is impossible to calculate the K value precisely and is not the focus of this paper, three percentages κ of Top$K$ are used in this paper, 1%, 20%, and 50%, corresponding to the first K large elements of the sparse dual gradient elements. The resilient adaptive penalty parameter method speeds up the convergence of the ADMM algorithm, and the Top$K$ sparsity in turn reduces the computational complexity from $\mathcal{O}(d*\ \sigma)$ to $\mathcal{O}(d*\ \sigma*\ \kappa)$.

5.3. Distributed Algorithm 2D-TGSA-TPADMM

To decrease the training time of the 2D-TGSA-ADMM algorithm, we propose a new algorithm called 2D-TGSA-TPADMM, which combines the resilient adaptive penalty parameter method with the Top$K$ sparse computation method. The algorithm is shown in Algorithm 3.

Algorithm 3: 2D-TGSA-TPADMM on worker i

input : The maximum iteration $\mathcal{K}$
Group numbers L
output: Global parameter z after iterations

initialize $\rho_i^0 = 1$, $w_i^0 = \mathbf{0}$, $x_i^0 = \mathbf{0}$, $\lambda_i^0 = \mathbf{0}$, $z^0 = \mathbf{0}$;
for $k \leftarrow 0$ **to** $\mathcal{K}$ **do**
  update x_i^{k+1} by (5);
  update w_i^{k+1} by (8);
  $\boldsymbol{w}^{k+1} \leftarrow$ 2D-TGSA(w_i^{k+1},L);
  update z_g^{k+1} by (6);
  for $j \leftarrow 0$ **to** $d * \sigma$ **do**
    Select threshold:$thr \leftarrow \kappa$ of $|G_i^k[j]|$;
    $Mask \leftarrow |G_i^k[j]| > thr$;
    $\tilde{G}_i^k \leftarrow G_i^k[j] \odot Mask$;
    $0 \leftarrow G_i^k[j] \odot \neg Mask$;
    update ρ_i^{k+1} by (13) ;
  update λ_i^{k+1} by (7);
  $k \leftarrow k+1$;
return $z^{\mathcal{K}}$;

5.3.1. Convergence Analysis

In this subsection, we provide convergence analysis for the proposed distributed algorithm 2D-TGSA-TPADMM.

Symbol description. For any vectors $(x_1, \ldots, x_N, z, \lambda) \in \mathbb{R}^d$, we let $(x_1, \ldots, x_N, z, \lambda)$ be (x, z, λ) and $f(x) = \sum_{i=1}^N f(x_i)$.

Assumption 1. Functions f_i and g are convex functions with variable derivatives.

Assumption 2. The unaugmented Lagrangian L_0 has a saddle point. Explicitly, there exist (x^*, z^*, y^*), not necessarily unique, for which

$$L_0(x^*, z^*, \lambda) \leqslant L_0(x^*, z^*, \lambda^*) \leqslant L_0(x, z, \lambda^*), \tag{15}$$

holds for all (x, z, λ).

Theorem 1. Under Assumptions 1 and 2, the ADMM iterates satisfy $r^k \to 0, z^{k+1} - z^k \to 0$ as $k \to +\infty$, then

$$\lim_{k \to +\infty} p^k = p^*.$$

Proof. As (x^*, z^*, λ^*) is the saddle point of the function L_0, there is $L_0(x^*, z^*, \lambda^*) \leqslant L_0(x^{k+1}, z^{k+1}, \lambda^*)$. Using $x^* - z^* = 0$ and setting $p^{k+1} = f(x^{k+1}) + g(z^{k+1}), p^* = f(x^*) + g(z^*)$, we have

$$p^* \leqslant p^{k+1} + (\lambda^*)^T r^{k+1}. \tag{16}$$

By definition, we know that x^{k+1} is the minimum value of $L_\rho(x, z^k, \lambda^k)$. Since f is a differentiable convex function, there is $0 \in \partial L_\rho(x^{k+1}, z^k, \lambda^k) = \partial f(x^{k+1}) + \lambda^k + \rho(x^{k+1} - z^k)$. Recall that $\lambda^{k+1} = \lambda^k + \rho r^{k+1}$, and hence $0 \in \partial L_\rho(x^{k+1}, z^k, \lambda^k) = \partial f(x^{k+1}) + \lambda^{k+1} + \rho(z^{k+1} - z^k)$. This shows that x^{k+1} is the minimum value of the function

$$f(x) + (\lambda^{k+1} + \rho(z^{k+1} - z^k))^T x. \tag{17}$$

Similarly, we can get that z^{k+1} is the minimum of function $g(z) - (\lambda^{k+1})^T z$. Combining the function with (17), we can get

$$\begin{aligned} &f(x^{k+1}) + (\lambda^{k+1} + \rho(z^{k+1} - z^k))^T x^{k+1} \\ &\leqslant f(x^*) + (\lambda^{k+1} + \rho(z^{k+1} - z^k))^T x^*, \end{aligned} \tag{18}$$

$$g(z^{k+1}) - (\lambda^{k+1})^T z^{k+1} \leqslant g(z^*) - (\lambda^{k+1})^T z^*. \tag{19}$$

Using $x^* - z^* = 0$ and combining (18) and (19), we have

$$\begin{aligned} &p^{k+1} - p^* \\ &\leqslant -(\lambda^{k+1})^T r^{k+1} - \rho(z^{k+1} - z^k)^T (r^{k+1} + z^{k+1} - z^*). \end{aligned} \tag{20}$$

Combining (16) and (20), we get

$$2(\lambda^{k+1} - \lambda^*)^T r^{k+1} + 2\rho(z^{k+1} - z^k)^T r^{k+1} + 2\rho(z^{k+1} - z^k)^T (z^{k+1} - z^*) \leqslant 0. \tag{21}$$

Using $\lambda^{k+1} = \lambda^k + \rho r^{k+1}$, the first term of (21) can be re-written as

$$\begin{aligned} 2(\lambda^{k+1} - \lambda^*)^T r^{k+1} &= 2(\lambda^k - \lambda^*)^T r^{k+1} + \rho\|r^{k+1}\|_2^2 + \rho\|r^{k+1}\|_2^2 \\ &= (2/\rho)(\lambda^k - \lambda^*)^T (\lambda^{k+1} - \lambda^k) \\ &\quad + (1/\rho)\|\lambda^{k+1} - \lambda^k\|_2^2 + \rho\|r^{k+1}\|_2^2 \\ &= (1/\rho)(\|\lambda^{k+1} - \lambda^*\|_2^2 - \|\lambda^k - \lambda^*\|_2^2) + \rho\|r^{k+1}\|_2^2. \end{aligned} \tag{22}$$

Re-arranging (22) and the remaining items of the left hand-side of (21)

$$\begin{aligned}
&\rho\|r^{k+1}\|_2^2 + 2\rho(z^{k+1}-z^k)^T r^{k+1} + 2\rho(z^{k+1}-z^k)^T(z^{k+1}-z^*)\\
&= \rho\|r^{k+1} + (z^{k+1}-z^k)\|_2^2 - \rho\|z^{k+1}-z^k\|_2^2 + 2\rho(z^{k+1}-z^k)^T(z^{k+1}-z^*)\\
&= \rho\|r^{k+1} + (z^{k+1}-z^k)\|_2^2 - \rho(\|(z^{k+1}-z^k)-(z^{k+1}-z^*)\|_2^2 - \|z^{k+1}-z^*\|_2^2)\\
&= \rho\|r^{k+1} + (z^{k+1}-z^k)\|_2^2 + \rho(\|z^{k+1}-z^*\|_2^2 - \|z^k-z^*\|_2^2).
\end{aligned} \tag{23}$$

Using $U_k = (1/\rho)\|\lambda^k - \lambda^*\|_2^2 + \rho\|z^k - z^*\|_2^2$, (21) can be written as

$$U_k - U_{k+1} \geqslant \rho\|r^{k+1} + (z^{k+1}-z^k)\|_2^2. \tag{24}$$

Since z^{k+1} is the minimum value of function $g(z) - (\lambda^{k+1})^T z$ and z^k is the minimum value of function $g(z) - (\lambda^k)^T z$, we can get

$$g(z^{k+1}) - (y^{k+1})^T z^{k+1} \leqslant g(z^k) - (y^{k+1})^T z^k, \tag{25}$$

$$g(z^k) - (y^k)^T z^k \leqslant g(z^{k+1}) - (y^k)^T z^{k+1}. \tag{26}$$

Combining the above two inequalities, there is

$$(\lambda^{k+1} - \lambda^k)^T(z^{k+1} - z^k) \geqslant 0. \tag{27}$$

Since $\lambda^{k+1} - \lambda^k = \rho r^{k+1}$, then we can write

$$\rho(r^{k+1})^T(z^{k+1} - z^k) \geqslant 0. \tag{28}$$

Combining (28) and (24), we get

$$U_k - U_{k+1} \geqslant \rho\|r^{k+1}\|_2^2 + \rho\|z^{k+1} - z^k\|_2^2. \tag{29}$$

So there is

$$\sum_{k=0}^{+\infty}(\rho\|r^{k+1}\|_2^2 + \rho\|z^{k+1} - z^k\|_2^2) \leqslant U_0. \tag{30}$$

From (30), we conclude that the ADMM iterates satisfy $r^k \to 0, z^{k+1} - z^k \to 0$ as $k \to +\infty$, then

$$\lim_{k\to+\infty} p^k = p^*.$$

6. Experiments

In this section, we evaluate the performance and scalability of the sparse synchronization algorithm 2D-TGSA and the proposed distributed algorithm 2D-TGSA-TPADMM.

Problem formulation. We use the general consensus ADMM algorithm to solve large-scale logistic regression problem with ℓ_1-norm, as shown in Eq. (31).

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \equiv \sum_{i=1}^{n} log(1 + e^{-b_i \boldsymbol{x}^T D_i}) + \frac{1}{2}\|\boldsymbol{x}\|_1, \tag{31}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is model parameters, n is the number of samples, $D_i \in \mathbb{R}^d$ is the i-th sample, $b_i \in \{-1, 1\}$ represents the label of the i-th sample.

6.1. Synchronization Test of Model Parameters

Fig. 5 shows a flow chart of the distributed ADMM algorithm used for training ML models. The algorithm for synchronizing model parameters involves each worker reducing a sub-model to a global model, as depicted in the orange section of Fig. 5. An effective synchronization algorithm can enhance the scalability of the distributed algorithm. To evaluate the performance of 2D-TGSA, we conduct experiments to compare it with three existing synchronization algorithms.

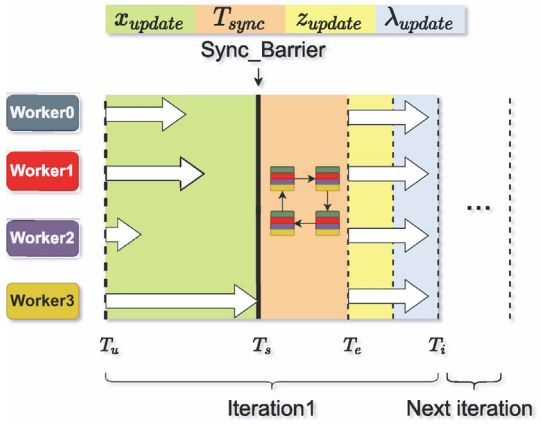


Fig. 5. Time breakdown of one iteration of the distributed algorithm. The y-axis represents different workers, and the x-axis denotes the time of each part.

6.1.1. Experimental setting

Evaluation mechanism. One iteration of the ADMM algorithm includes the update times for primal variable x_{update}, consistency variable z_{update}, dual variable λ_{update} and sub-model synchronization T_{sync}, as illustrated in Fig. 5. The sub-model synchronization time is used to evaluate the performance of the synchronization algorithm.

Datasets. In order to evaluate the efficiency of the proposed sparse synchronization algorithm, we test two high-dimensional sparse datasets, namely `kddbr`[1] and `url`[2], as shown in Table 3. `kddbr` is a dataset provided by the 2010 KDD Cup competition, while `url` is a binary dataset used to classify normal and malicious web pages. The provided datasets are in the LIBSVM format.

Experimental platform configuration information. The configuration information of our experimental platform is shown in Table 4, and the configuration information of the cluster is shown in Table 5.

Baseline. We compare 2D-TGSA to the following state-of-the-art algorithms:

(1) WRHT [42].
(2) Moshpit AllReduce [43].
(3) HSA [15].

6.1.2. Performance and Scalability

To compare the synchronization efficiency of the algorithms, we conduct experiments with different scenarios using 16 and 64 workers, respectively. Table 6 presents all the experimental results. In most cases, the 2D-TGSA algorithm exhibits better synchronization efficiency than all other algorithms tested on benchmark datasets under different schemes.

Synchronization time. We employ an ADMM-based distributed algorithm to train the logistic regression model on sparse datasets and measure synchronization time as a test metric. Table 6 summarizes the synchronization time for each round of communication. We provide a more detailed analysis of the relative performance of different message sizes and model sparse rates. Firstly, we conduct experiments on the `kddbr` dataset, and the results are shown in Fig. 6. Our findings indicate that the 2D-TGSA algorithm consistently outperforms WRHT, Moshpit AllReduce, and HSA algorithms by 2.4×, 5.1×, and 1.3×, respectively, when the

[1] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html/##kdd2010 raw version (bridge to algebra)
[2] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html##url

Table 3
A summary of datasets.

Dataset	#Training instances	#Testing instances	#Features
kddbr	19,264,097	748,401	1,163,024
url	2,156,517	239,613	3,231,961

Table 4
Platform configuration information.

Platform configuration	Parameter value
#CPU	1 × Intel(R) Core(TM) i7-10700 CPU @ 2.90 GHz
#Cores per CPU	8
RAM per computer	16 GB
Operating system	Ubuntu 20.04

Table 5
Cluster configuration information.

Cluster configuration	Parameter value
#Nodes in the cluster	10
#Cores in the cluster	80
MPI version	MPICH-3.3a2
Compiler	gcc-7.5.0
Network	Giga Bit Ethernet

model parameter sparse rate is 85.68%. As the computational scale increases, the model parameter sparsity increases to 92.84%, and the synchronization efficiency increases by 2.3×, 6.4×, and 1.5×, respectively. In addition, we conduct experiments on the `url` dataset, and the results are shown in Fig. 7. Our observations indicate that the 2D-TGSA algorithm consistently outperforms WRHT, Moshpit AllReduce, and HSA algorithms by 2.1×, 5×, and 1.1×, respectively, when the model parameter sparse rate is 86.26%. As the computational scale increases, the model parameter sparsity increases to 94.72%, and the synchronization efficiency increases by 2.6×, 7.5×, and 1.6×, respectively.

The experimental results are consistent with the theoretical analysis presented in Section 5.1.4, which demonstrates the effectiveness of our sparse synchronization algorithm in a small-scale distributed cluster. These findings also indicate that the 2D-TGSA algorithm is highly effective in improving performance when dealing with high-dimensional sparse datasets. On the other hand, communication sparseness is necessary, particularly in heterogeneous environments and with limited bandwidth at the edges.

Table 6
The synchronization time of various algorithms.

Dataset	#Workers	Model parameter sparse rate	Synchronization algorithm	Synchronization time per round (s)
kddbr	16	85.68%	WRHT	0.50∼0.57
			Moshpit AllReduce	1.01∼1.27
			HSA	0.26∼0.32
			2D-TGSA	0.20∼0.25
	64	92.84%	WRHT	0.78∼0.81
			Moshpit AllReduce	2.06∼2.32
			HSA	0.49∼0.53
			2D-TGSA	0.32∼0.36
url	16	86.26%	WRHT	1.42∼1.46
			Moshpit AllReduce	3.10∼3.61
			HSA	0.71∼0.76
			2D-TGSA	0.65∼0.69
	64	94.72%	WRHT	2.19∼2.22
			Moshpit AllReduce	6.21∼6.72
			HSA	1.30∼1.43
			2D-TGSA	0.75∼0.98

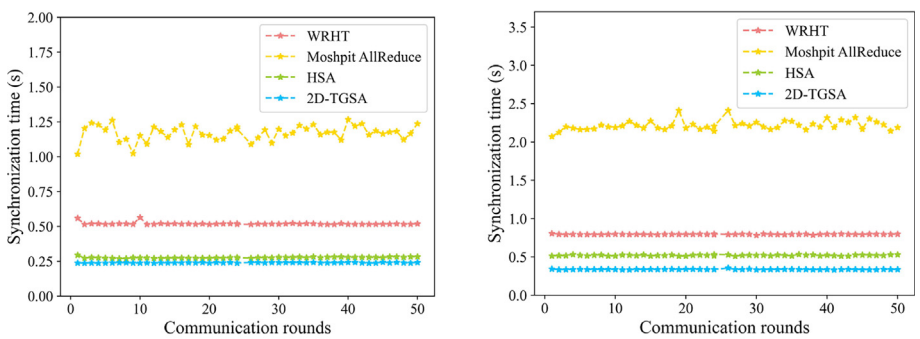


Fig. 6. The synchronization time of four various synchronization algorithms using the `kddbr` dataset with 16 and 64 workers.

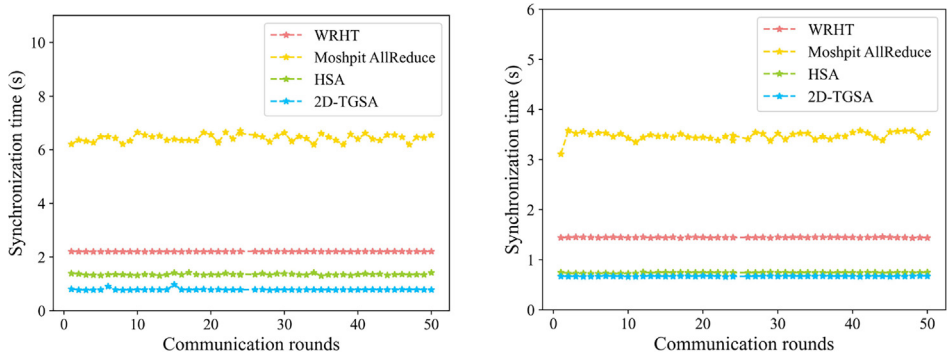


Fig. 7. The synchronization time of four various synchronization algorithms using the `url` dataset with 16 and 64 workers.

Table 7
A summary of high-dimensional datasets.

Dataset	#Training instances	#Testing instances	#Features
avazu	12,642,186	1,719,304	1,000,000
webspam	280,000	70,000	16,609,143
kdd2012	119,705,032	29,934,073	54,686,452

Table 8
Tianhe-2 platform configuration information.

Platform configuration	Parameter value
#CPUs per computer	2 × Intel Xeon E5
#Cores per CPU	12
RAM per computer	64 GB
Operating system	Red Hat Enterprise Linux Server release 6.5

Table 9
Tianhe-2 MPI cluster configuration information.

Cluster configuration	Parameter value
#Nodes in the cluster	64
#Cores in the cluster	1536
MPI version	MPICH-3.1
Compiler	GNU-4.8.4
Network	TH2 Express-2 + 14 Gbps × 8lane

6.2. Accuracy and Efficiency of 2D-TGSA-TPADMM

6.2.1. Experiment Setup

Datasets. Three high-dimensional sparse datasets, `avazu`[3] `webspam`[4] and `kdd2012`[5], are chosen for the experiments. The `avazu` dataset is used for predicting click-through rates, while the `webspam` dataset is intended for research on web spam detection. The `kdd2012` dataset is provided by the 2012 KDD Cup competition and is used for predicting the click-through rate of ads based on query and user information. All three datasets are binary and are in the LIBSVM format. Detailed descriptions of the datasets are provided in Table 7.

Experimental platform configuration information. The configuration information of our experimental platform is shown in Table 8, and the cluster configuration information is shown in Table 9.

Hyper-parameters. For all experiments, the ℓ_1-norm and the initial penalty parameter are set to 0.5 and 1, respectively. The maximum number of iterations and conjugate gradient parameter of the TRON are 1000 and 0.1, respectively.

Implementation. All methods are implemented in *c++*, and the communication library uses MPICH. All the steps of the implemented method run on the CPU except loading the training data from the disk and communicating the model parameters through the network.

6.2.2. Effect of κ

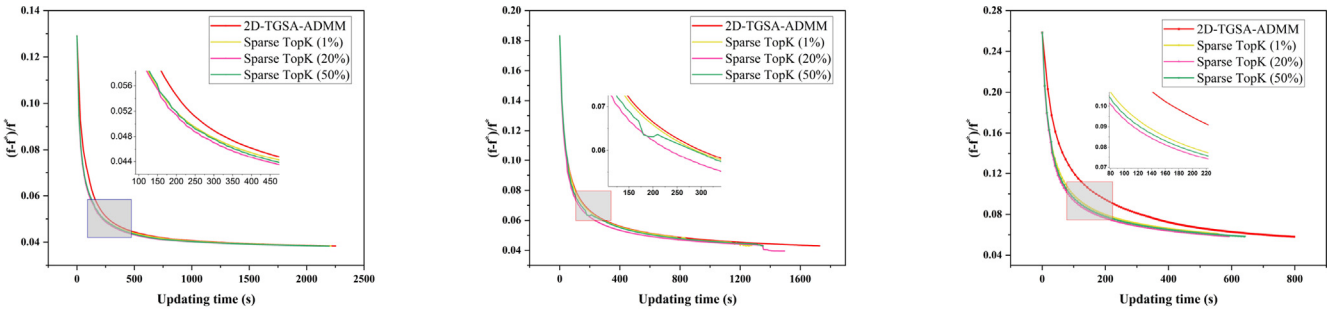
We investigate the effect of the choice of K on the performance of the 2D-TGSA-TPADMM. The sparse Top$K$ method for dual variables does not require an exact value of K. Instead, we choose the proportion κ of the maximum gradient elements of the dual variable λ as the K value for the sparse Top$K$ computation. The proportions are set to $1\%, 20\%$ and 50%, and we test the optimal κ on the Tianhe-2 supercomputing platform using the sparse `webspam` dataset.

In order to test the scalability of the algorithm, we use three testing schemes with 2, 4, and 18 nodes consisting of 16, 64, and 256 workers, respectively, as shown in Fig. 8. The experimental results reveal that the use of Top$K$ sparse computation could speed up the training speed of the 2D-TGSA-AMMM and improve the model's testing accuracy. With an increase in the number of workers, the parameters of the model become sparser, resulting in better performance of 2D-TGSA-TPADMM. Regarding the choice of sparse proportion, we find that selecting 20% as the dual gradient Top$K$ sparse proportion works better than the other tested proportions, as shown in Fig. 8. Therefore, in the following experiments, our sparse Top$K$ proportion κ is set to 20%.

6.2.3. Experimental Results

Baseline. We compare 2D-TGSA-TPADMM to the following state-of-the-art distributed algorithms:

[3] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html/##avazu
[4] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html/##webspam
[5] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html/##kdd2012



(a) 16 workers. (b) 64 workers. (c) 256 workers.

Fig. 8. Comparison of convergence performance of 2D-TGSA-ADMM and 2D-TGSA-TPADMM with different sparse Top$K$ proportions of 1%, 20%, and 50%.

(1) LBFGS-ADMM [40]. Communication Efficient quasi-Newton Method based on the Alternating Direction Method of Multipliers.
(2) HSAC-ALADMM [15]. Global Consistent ADMM Based on Hierarchical Sparse AllReduce.
(3) 2D-TGSA-ADMM (proposed).

Comparison to baselines. We conduct experiments under different scenarios, using 16, 64, and 256 workers, respectively, to compare the performance of the methods. Due to differences in dataset sizes and model dimensions, we use 2, 4, and 16 compute nodes to test the `avazu` dataset, 2, 4, and 18 compute nodes to test the `webspam` dataset, and 3, 10, and 32 compute nodes to test the `kdd2012` dataset. For all experimental results shown in Table 10, 2D-TGSA-TPADMM outperforms all other scenarios in the three datasets.

Synchronization time. We evaluate the effectiveness of our proposed algorithm in an HPC environment by calculating the synchronization time of various ADMM-based distributed algorithms. Table 10 summarizes the synchronization time of each communication round.

In the `avazu` dataset, as the number of workers increases from 16 to 64 and 256, the sparsity of model parameters increases from 86.32% to 96.58% and 99.15%, respectively. Compared to the LBFGS-ADMM algorithm that uses the AllReduce synchronization algorithm, the 2D-TGSA-ADMM and 2D-TGSA-TPADMM algorithms based on 2D-TGSA show a 3.7×, 4.4×, and 2× improvement in the synchronization efficiency, respectively. Compared to the HSAC-ALADMM algorithm, which utilizes a hierarchical sparse AllReduce synchronization algorithm, the synchronization efficiency is improved by 2.5×, 3×, and 1.5×, respectively. In the `webspam` dataset, the sparsity of model parameters increases from 97.68% to 98.47% and 99.11%, respectively, as the number of workers increases from 16 to 64 and 256. The 2D-TGSA-ADMM and 2D-TGSA-TPADMM algorithms improve the synchronization efficiency by 3.1×, 3.2×, and 3.5×, respectively, compared with the LBFGS-ADMM algorithm. Compared to HSAC-ALADMM algorithm, the synchronization efficiency is improved by 2.1×, 2.3×, and 2.6×, respectively. In `kdd2012`, the feature dimension is larger than the other two datasets, and the trained model is denser than that of the `webspam` dataset. Compared to the LBFGS-ADMM algorithm, the synchronization efficiency is improved by 1.5×, 2.2×, and 3.8× respectively.

Overall, the results indicate that the 2D-TGSA-based distributed algorithms significantly improve the synchronization efficiency compared to the LBFGS-ADMM algorithm using the AllReduce synchronization algorithm. Furthermore, the proposed ADMM-based distributed algorithms outperform the HSAC-ALADMM algorithm, which utilizes a hierarchical sparse AllReduce synchronization algorithm. The experiments also suggest that the synchronization efficiency of the 2D-TGSA algorithm improves as the sparsity of the model parameters increases. The superior performance also demonstrates the effectiveness of our proposed grouped sparse AllReduce algorithm based on 2D-Torus topology.

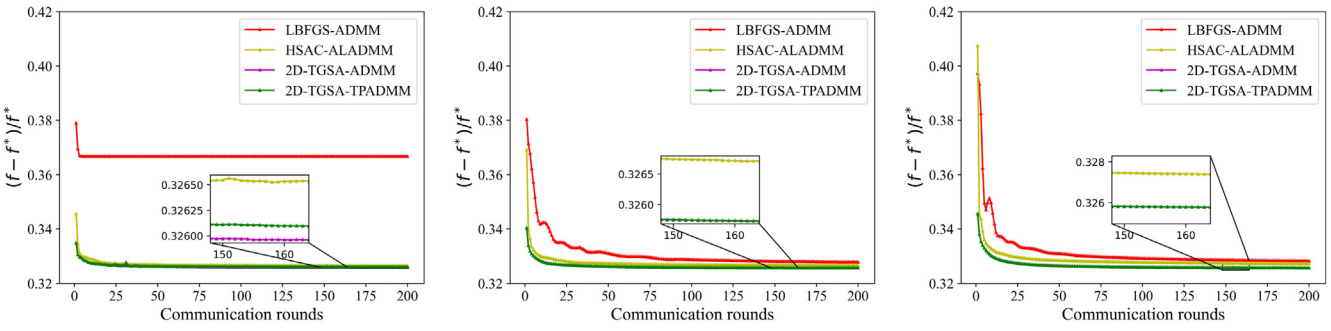
Updating time. The second-order algorithm is widely used to solve sub-problems in distributed ADMM algorithms due to its fast convergence rate. However, its high computational complexity often leads to inconsistent computing speed of nodes, and synchronization of model parameters requires workers to wait for each other, which decreases the efficiency of distributed algorithms. To reduce the computational complexity, we propose a sparse computation method that uses the average updating time of all workers as the evaluation metric. In our experiments on the `avazu`, `webspam`, and `kdd2012` datasets, we compare the performance of our proposed 2D-TGSA-ADMM and 2D-TGSA-TPADMM algorithms with two baseline algorithms, LBFGS-ADMM and HSAC-ALADMM. Our experimental results show that the 2D-TGSA-ADMM and 2D-TGSA-TPADMM algorithms not only take less updating time but also have a higher test accuracy than LBFGS-ADMM on all the three datasets. In addition, the computational efficiency of the proposed algorithms is significantly improved compared to HSAC-ALADMM.

As shown in Fig. 12(a) compared to HSAC-ALADMM, the 2D-TGSA-ADMM and 2D-TGSA-TPADMM algorithms achieve higher computational efficiency at 16 workers (↑ 13.13%, ↑ 30.05%), 64 workers (↑ 13.12%, ↑ 39.6%), and 256 workers (↑ 12.79%, ↑ 14.01%) in the `avazu` dataset. Similarly, as shown in Fig. 12(b), the computational efficiency of these algorithms is improved at 16 workers (↑ 11.77%, ↑ 19.06%), 64 workers (↑ 5.82%, ↑ 17.88%), and 256 workers (↑ 4.02%, ↑ 18.69%) in the `webspam` dataset. Finally, in the `kdd2012` dataset, as shown in Fig. 12(c), the computational efficiency is improved at 16 workers (↑ 41.41%, ↑ 45.46%), 64 workers (↑ 19.53%, ↑ 29.3%), and 256 workers (↑ 19.23%, ↑ 19.61%) compared to HSAC-ALADMM. These results demonstrate that the proposed algorithms are more efficient than HSAC-ALADMM for solving the logistic regression problem with ℓ_1-norm regularization on different datasets. Furthermore, as the size of the model increases, the computational efficiency of the proposed algorithm gradually improves, indicating the effectiveness of the proposed sparse computation method in reducing computational complexity.

Overall, we find that the 2D-TGSA-TPADMM algorithm strikes the best balance between performance and convergence speed. The superior performance and convergence speed of 2D-TGSA-TPADMM also demonstrate the effectiveness of our proposed resilient adaptive penalty parameter method and Top$K$ sparse computation scheme.

Table 10
Comparison of updating and synchronization times for various ADMM-based distributed algorithms across different sparse rates of model parameters tested on Tianhe-2.

Dataset	#Workers	Model parameter sparse rate	Distributed algorithm	Updating time (s)	Synchronization time per round (s)
avazu	16	86.32%	LBFGS-ADMM	1322.42	0.02∼0.04
			HSAC-ALADMM	859.04	0.01∼0.03
			2D-TGSA-ADMM	746.25	0.007∼0.01
			2D-TGSA-TPADMM	600.86	0.007∼0.01
	64	96.58%	LBFGS-ADMM	567.34	0.03∼0.05
			HSAC-ALADMM	324.92	0.02∼0.04
			2D-TGSA-ADMM	282.26	0.009∼0.01
			2D-TGSA-TPADMM	196.24	0.009∼0.01
	256	99.15%	LBFGS-ADMM	331.88	0.03∼0.05
			HSAC-ALADMM	184.65	0.02∼0.04
			2D-TGSA-ADMM	161.02	0.01∼0.03
			2D-TGSA-TPADMM	158.77	0.01∼0.03
webspam	16	97.68%	LBFGS-ADMM	5943.54	0.46∼0.48
			HSAC-ALADMM	2537.95	0.31∼0.34
			2D-TGSA-ADMM	2239.12	0.15∼0.16
			2D-TGSA-TPADMM	2054.14	0.15∼0.16
	64	98.47%	LBFGS-ADMM	4878.32	0.61∼0.63
			HSAC-ALADMM	1732.04	0.43∼0.45
			2D-TGSA-ADMM	1631.11	0.19∼0.20
			2D-TGSA-TPADMM	1422.3	0.19∼0.20
	256	99.11%	LBFGS-ADMM	4318.58	0.62∼0.64
			HSAC-ALADMM	1133.08	0.46∼0.48
			2D-TGSA-ADMM	1087.44	0.17∼0.19
			2D-TGSA-TPADMM	921.29	0.17∼0.19
kdd2012	16	89.81%	LBFGS-ADMM	15640.05	1.53∼1.55
			HSAC-ALADMM	21384.6	1.14∼1.16
			2D-TGSA-ADMM	12528.78	1.03∼1.05
			2D-TGSA-TPADMM	11662.2	1.03∼1.05
	64	96.46%	LBFGS-ADMM	7453	1.68∼1.70
			HSAC-ALADMM	9595.37	0.76∼0.79
			2D-TGSA-ADMM	7720.75	0.75∼0.77
			2D-TGSA-TPADMM	6784.07	0.75∼0.77
	256	98.78%	LBFGS-ADMM	9325.24	2.82∼2.87
			HSAC-ALADMM	6342.06	0.75∼0.78
			2D-TGSA-ADMM	5122.06	0.73∼0.75
			2D-TGSA-TPADMM	5098.09	0.73∼0.75



(a) 16 workers. (b) 64 workers. (c) 256 workers.

Fig. 9. Comparison of convergence performance of various ADMM-based distributed algorithms on the `avazu` dataset.

Convergence performance and test accuracy. To verify the superiority of the 2D-TGSA-TPADMM algorithm, we compare its convergence performance with that of the LBFGS-ADMM and HSAC-ALADMM algorithms in this paper. We run each algorithm 200 times and obtain the iterative curves in Fig. 9–11 by averaging. From the three figures, we can see that all the four algorithms can converge to the objective value. The 2D-TGSA-TPADMM algorithm converges the quickest, followed by the 2D-TGSA-ADMM, HSAC-ALADMM and LBFGS-ADMM algorithms.

Specifically, the test accuracy of 2D-TGSA-ADMM and 2D-TGSA-TPADMM is higher than that of the baselines, as shown in Fig. 12. The main reason is that the algorithm proposed in this paper uses a general form consensus ADMM algorithm. The local variable no longer corresponds to the global variable z but to a part of the global variable z_g. By training logistic regression models with ℓ_1-norm on the three datasets, we find that the 2D-TGSA-TPADMM algorithm works better for high-dimensional sparse models, which shows that our sparse algorithm is a good solution for high-dimensional sparse datasets.

7. Conclusions and Future Work

In order to enable large-scale machine learning, we develop, analyze, and evaluate an ADMM algorithm framework with high communication efficiency in a distributed environment.

Our proposed distributed algorithm, 2D-TGSA-TPADMM, uses a duality approach to derive sub-problems for each machine to solve in parallel. These sub-problems closely match the global problems, which makes it easy to reuse the most advanced single-machine

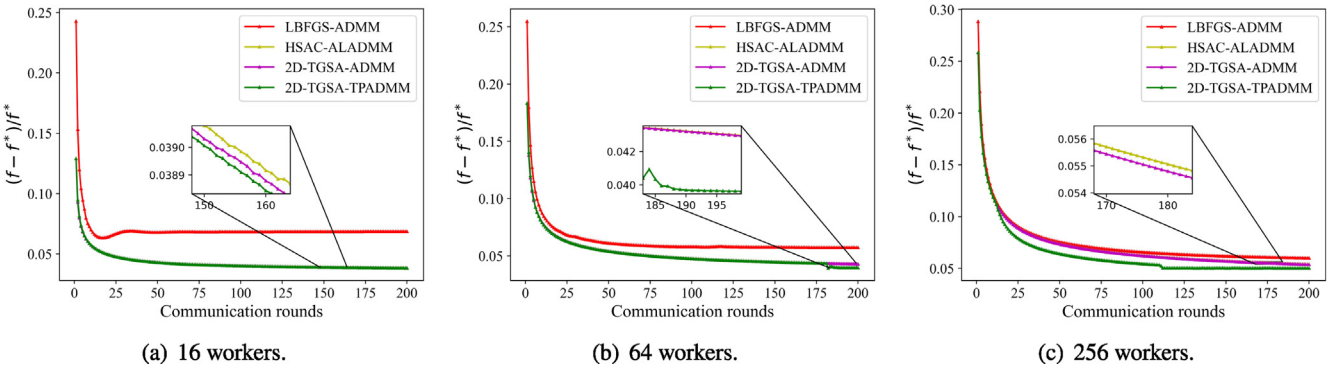


Fig. 10. Comparison of convergence performance of various ADMM-based distributed algorithms on the `webspam` dataset.

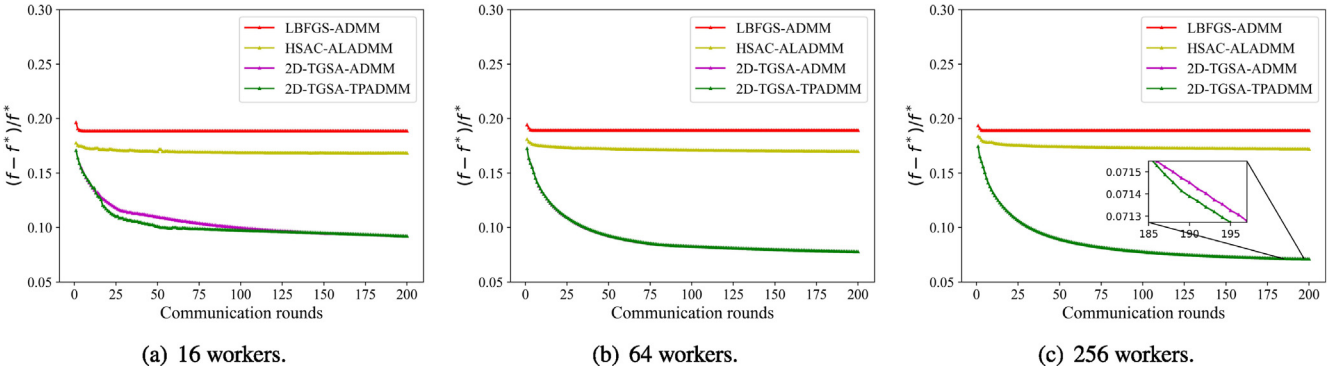


Fig. 11. Comparison of convergence performance of various ADMM-based distributed algorithms on the `kdd2012` dataset.

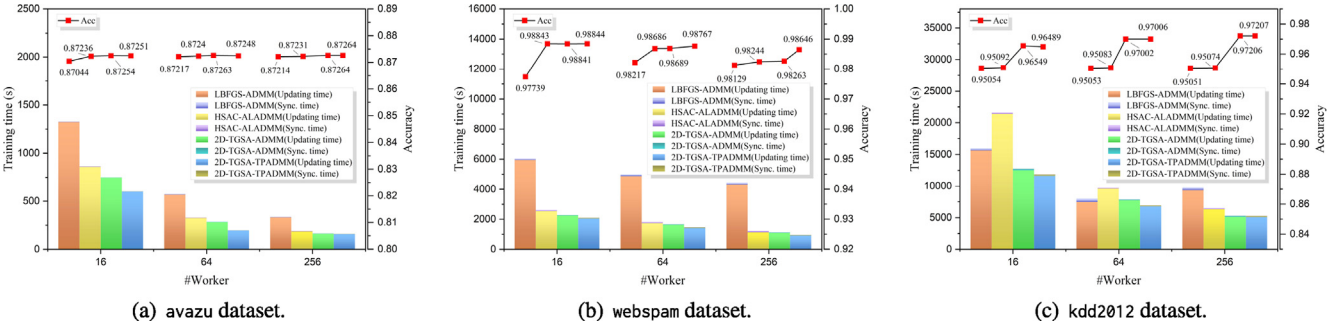


Fig. 12. Scalability comparison of various ADMM-based distributed algorithms in `avazu`, `webspam` and `kdd2012` datasets with #Worker increases from 16 to 256.

solver in the distributed environment. Furthermore, the distributed algorithm supports a highly flexible communication mode. Especially, the local solvers update their local parameters directly, which reduces the need for communication, and the proposed 2D-TGSA can help to improve communication efficiency in the distributed environment. We reduce computational complexity and adaptively adjust penalty parameters at every iteration to achieve faster training speeds while ensuring convergence guarantees. Finally, we demonstrate the efficiency of our algorithm in an extensive experimental comparison with state-of-the-art distributed algorithms. Our algorithm outperforms other widely used methods on real-world distributed datasets.

A single synchronization method cannot meet the diverse requirements of distributed training environments. Choosing an appropriate synchronization method is therefore meaningful for different distributed training environments. Future work will focus on how to adaptively choose proper synchronization modes according to different cluster environments during training, instead of choosing only one synchronization method.

CRediT authorship contribution statement

Guozheng Wang: Conceptualization, Methodology, Software, Formal analysis, Visualization, Writing - original draft. **Yongmei Lei:** Writing - review & editing, Funding acquisition, Project administration. **Yongwen Qiu:** Data curation, Supervision. **Lingfei Lou:** Validation, Resources. **Yixin Li:** Formal analysis, Resources.

Data availability

Data will be made available on request.

Declaration of Competing Interest

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

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grant No.U1811461).

References

[1] M. Jena, R.K. Behera, S. Dehuri, Hybrid decision tree for machine learning: A big data perspective, in: Advances in Machine Learning for Big Data Analysis, Springer, 2022, pp. 223–239.
[2] M. Shehab, L. Abualigah, Q. Shambour, M.A. Abu-Hashem, M.K.Y. Shambour, A. I. Alsalibi, A.H. Gandomi, Machine learning in medical applications: A review of state-of-the-art methods, Computers in Biology and Medicine 145 (2022).
[3] F. Noé, A. Tkatchenko, K.-R. Müller, C. Clementi, Machine learning for molecular simulation, Annual review of physical chemistry 71 (2020) 361–390.
[4] L. Kong, W. He, Y. Dong, L. Cheng, C. Yang, Z. Li, Asymmetric bounded neural control for an uncertain robot by state feedback and output feedback, IEEE Transactions on Systems, Man, and Cybernetics: Systems 51 (3) (2019) 1735–1746.
[5] L. Kong, W. He, W. Yang, Q. Li, O. Kaynak, Fuzzy approximation-based finite-time control for a robot with actuator saturation under time-varying constraints of work space, IEEE transactions on cybernetics 51 (10) (2020) 4873–4884.
[6] T.T. Nguyen, M. Wahib, R. Takano, Topology-aware sparse allreduce for large-scale deep learning, in: 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), IEEE, 2019, pp. 1–8.
[7] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J.S. Rellermeyer, A survey on distributed machine learning, ACM Computing Surveys (CSUR) 53 (2) (2020) 1–33.
[8] Q. Tong, G. Liang, X. Cai, C. Zhu, J. Bi, Asynchronous parallel stochastic quasi-newton methods, Parallel Computing 101 (2021).
[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends in, Machine Learning 3 (1) (2011) 1–122.
[10] C.-H. Fang, S.B. Kylasa, F. Roosta, M.W. Mahoney, A. Grama, Newton-admm: A distributed gpu-accelerated optimizer for multiclass classification problems, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–12.
[11] S.W. Fung, S. Tyrväinen, L. Ruthotto, E. Haber, Admm-softmax: an admm approach for multinomial logistic regression, arXiv preprint arXiv:1901.09450 (2019).
[12] Z. Zhang, S. Yang, W. Xu, K. Di, Privacy-preserving distributed admm with event-triggered communication, IEEE Transactions on Neural Networks and Learning Systems (2022) 1–13.
[13] L. Kong, W. He, Z. Liu, X. Yu, C. Silvestre, Adaptive tracking control with global performance for output-constrained mimo nonlinear systems, IEEE Transactions on Automatic Control (2022) 1–8.
[14] J. Xie, Y. Lei, Admmlib: a library of communication-efficient ad-admm for distributed machine learning, in: IFIP International Conference on Network and Parallel Computing, Springer, 2019, pp. 322–326.
[15] D. Wang, Y. Lei, J. Xie, G. Wang, Hsac-aladmm: an asynchronous lazy admm algorithm based on hierarchical sparse allreduce communication, The Journal of Supercomputing 77 (8) (2021) 8111–8134.
[16] C.B. Issaid, A. Elgabli, J. Park, M. Bennis, M. Debbah, Communication efficient distributed learning with censored, quantized, and generalized group admm, arXiv preprint arXiv:2009.06459 (2020).
[17] Y. Liu, G. Wu, Z. Tian, Q. Ling, Dqc-admm: Decentralized dynamic admm with quantized and censored communications, IEEE Transactions on Neural Networks and Learning Systems 33 (8) (2022) 3290–3304.
[18] Z. Cai, X. Yan, K. Ma, Y. Wu, Y. Huang, J. Cheng, T. Su, F. Yu, Tensoropt: Exploring the tradeoffs in distributed dnn training with auto-parallelism, IEEE Transactions on Parallel and Distributed Systems 33 (8) (2021) 1967–1981.
[19] B. Yuan, C.R. Wolfe, C. Dun, Y. Tang, A. Kyrillidis, C. Jermaine, Distributed learning of fully connected neural networks using independent subnet training, Proceedings of the VLDB Endowment 15 (8) (2022) 1581–1590.
[20] J. Zerwas, K. Aykurt, S. Schmid, A. Blenk, Network traffic characteristics of machine learning frameworks under the microscope, in: 2021 17th International Conference on Network and Service Management (CNSM), IEEE, 2021, pp. 207–215.
[21] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, B.-Y. Su, Scaling distributed machine learning with the parameter server, in: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 2014, pp. 583–598.
[22] X. Miao, Y. Shi, H. Zhang, X. Zhang, X. Nie, Z. Yang, B. Cui, Het-gmp: a graph-based system approach to scaling large embedding model training, in: Proceedings of the 2022 International Conference on Management of Data, 2022, pp. 470–480.
[23] S. Dong, X. Miao, P. Liu, X. Wang, B. Cui, J. Li, Het-kg: Communication-efficient knowledge graph embedding training via hotness-aware cache, in: 2022 IEEE 38th International Conference on Data Engineering (ICDE), IEEE, 2022, pp. 1754–1766.
[24] A. Sergeev, M. Del Balso, Horovod: fast and easy distributed deep learning in tensorflow, arXiv preprint arXiv:1802.05799 (2018).
[25] X. Miao, X. Nie, Y. Shao, Z. Yang, J. Jiang, L. Ma, B. Cui, Heterogeneity-aware distributed machine learning training via partial reduce, in: Proceedings of the 2021 International Conference on Management of Data, 2021, pp. 2262–2270.
[26] A. Gibiansky, Bringing hpc techniques to deep learning, http://research.baidu.com/bringing-hpc-techniques-deep-learning, [Online; accessed 6-December-2017] (2017).
[27] J. Huang, P. Majumder, S. Kim, A. Muzahid, K.H. Yum, E.J. Kim, Communication algorithm-architecture co-design for distributed deep learning, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), IEEE, 2021, pp. 181–194.
[28] S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, X. Chu, A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2019, pp. 2238–2247.
[29] S. Shi, K. Zhao, Q. Wang, Z. Tang, X. Chu, A convergence analysis of distributed sgd with communication-efficient gradient sparsification, in: IJCAI, 2019, pp. 3411–3417.
[30] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, T. Hoefler, Sparcml: High-performance sparse communication for machine learning, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019, pp. 1–15.
[31] C.-Y. Chen, J. Ni, S. Lu, X. Cui, P.-Y. Chen, X. Sun, N. Wang, S. Venkataramani, V. V. Srinivasan, W. Zhang, et al., Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training, Advances in Neural Information Processing Systems 33 (2020) 13551–13563.
[32] J. Fei, C.-Y. Ho, A.N. Sahu, M. Canini, A. Sapio, Efficient sparse collective communication and its application to accelerate distributed deep learning, in: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, 2021, pp. 676–691.
[33] S. Zhou, G.Y. Li, Communication-efficient admm-based federated learning, arXiv preprint arXiv:2110.15318 (2021).
[34] S. Zhou, G.Y. Li, Federated learning via inexact admm, arXiv preprint arXiv:2204.10607 (2022).
[35] R. Bao, X. Wu, W. Xian, H. Huang, Doubly sparse asynchronous learning for stochastic composite optimization, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI, 2022, pp. 1916–1922.
[36] R. Bao, B. Gu, H. Huang, Fast oscar and owl regression via safe screening rules, in: in: International Conference on Machine Learning, PMLR, 2020, pp. 653–663.
[37] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[38] T. Zhao, X. Song, J. Li, W. Luo, I. Razzak, Distributed optimization of graph convolutional network using subgraph variance, arXiv preprint arXiv:2110.02987 (2021).
[39] Q. Ye, Y. Zhou, M. Shi, J. Lv, Flsgd: free local sgd with parallel synchronization, The Journal of Supercomputing 78 (10) (2022) 12410–12433.
[40] Y. Li, P.G. Voulgaris, N.M. Freris, A communication efficient quasi-newton method for large-scale distributed multi-agent optimization, in: ICASSP 2022–2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022, pp. 4268–4272.
[41] C.-J. Lin, R.C. Weng, S.S. Keerthi, Trust region newton methods for large-scale logistic regression, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 561–568.
[42] F. Dai, Y. Chen, Z. Huang, H. Zhang, F. Zhang, Efficient all-reduce for distributed dnn training in optical interconnect systems, in: Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, 2023, pp. 422–424.
[43] M. Ryabinin, E. Gorbunov, V. Plokhotnyuk, G. Pekhimenko, Moshpit sgd: Communication-efficient decentralized training on heterogeneous unreliable devices, Advances in Neural Information Processing Systems 34 (2021) 18195–18211.
[44] B.S. He, H. Yang, S.L. Wang, Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities, Journal of Optimization Theory and Applications 106 (2) (2000) 337–356.
[45] Z. Xu, M. Figueiredo, T. Goldstein, Adaptive admm with spectral penalty parameter selection, in, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 718–727.
[46] S. Khirirat, S. Magnússon, A. Aytekin, M. Johansson, A flexible framework for communication-efficient machine learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 8101–8109.

Guozheng Wang received the MS degree from the University of Shanghai for Science and Technology, China, in 2019, where he is currently working toward the PhD degree in the Shanghai University. His current research interests include the area of big data, High-performance computing and distributed optimization algorithms.

Yongmei Lei received the PhD degree from the College of Science, Xi'an Jiaotong University, Xian, China. She is currently a full professor in the School of Computer Engineering and Science, Shanghai University, Shanghai, China. Her research interests mainly focus on research and development of computer system architecture, high-performance computer development, and cross-disciplinary applications. She has published more than 50 papers in refereed international conferences and journals.

Yongwen Qiu received the BS degree from Shanghai University in 2021 and is now pursuing his MS degree in the School of Computer Engineering and Science at Shanghai University. His main research interests are high-performance computing and parallel computing.

Lingfei Lou received the BS degree from Shanghai Maritime University, Shanghai, China, and now is working toward the MS degree at Shanghai University. Her research interests mainly focus on big data, distributed computing, and High-performance computing.

Yixin Li received the BS degree from the University of Shanghai for Science and Technology, Shanghai, China, and now pursues the MS degree in the School of Computer Engineering and Science at Shanghai University. Her research interests mainly focus on distributed algorithms and parallel computing.