Computing and Informatics, Vol. 38, 2019, 765–789, doi: 10.31577/cai_2019_4_765

GROUP-BASED ASYNCHRONOUS DISTRIBUTED ALTERNATING DIRECTION METHOD OF MULTIPLIERS IN MULTICORE CLUSTER

Dongxia Wang, Yongmei Lei*, Shenghong Jiang

School of Computer Engineering and Science
Shanghai University
Nanchen Road No. 333, Baoshan District, Shanghai, 200436, China
e-mail: `wangdongxia1983@126.com, lei@shu.edu.cn, jsh79@i.shu.edu.cn`

Abstract. The distributed alternating direction method of multipliers (ADMM) algorithm is one of the effective methods to solve the global consensus optimization problem. Considering the differences between the communication of intra-nodes and inter-nodes in multicore cluster, we propose a group-based asynchronous distributed ADMM (GAD-ADMM) algorithm: based on the traditional star topology network, the grouping layer is added. The workers are grouped according to the process allocation in nodes and model similarity of datasets, and the group local variables are used to replace the local variables to compute the global variable. The algorithm improves the communication efficiency of the system by reducing communication between nodes and accelerates the convergence speed by relaxing the global consistency constraint. Finally, the algorithm is used to solve the logistic regression problem in a multicore cluster. The experiments on the Ziqiang 4000 showed that the GAD-ADMM reduces the system time cost by 35 % compared with the AD-ADMM.

Keywords: ADMM, global consensus optimization, multicore cluster, logistic regression, GAD-ADMM

Mathematics Subject Classification 2010: 68W15

* Corresponding author

1 INTRODUCTION

Machine learning has become an important method to extract structured information from raw data and transform it into different automatic predictions and applied hypotheses [1]. In the era of big data, sometimes not only is the number of samples large, but also the dimension of samples is high. Therefore, in the traditional machine learning algorithm, it is difficult to implement the relevant processing and calculation of the big data in a reasonable amount of time. It is necessary to consider how to transform the traditional machine learning algorithms into distributed ones using high-performance distributed computing. Most supervised machine learning algorithms can be viewed as cost-function optimization methods [1], which can be expressed as the following mathematical form:

$$\min f_x(x, D) \tag{1}$$

where $D \in R^{m*n}$ is the sample dataset, $x \in R^n$ represents the model parameter, m is the number of samples and n is the dimension of the samples.

The alternating direction method of multipliers (ADMM) is an effective method suitable for separable convex optimization, which has been used in distributed optimization and statistical machine learning [2]. The ADMM algorithm can transform the large global problem into several smaller, local sub-problems, and can derive the solution of the global problem by coordinating the solutions of the sub-problems [2]. We can transform the original problem (1) into a global consensus optimization problem, suitable for distributed environments, as shown below:

$$\min \sum_{i=1}^{N} f_i(x_i, D_i) + g(z), \quad \text{s.t. } x_i = z, i = 1, 2, \ldots, N \tag{2}$$

where $D_i \in R^{m_i*n}$, $\sum_{i=1}^{N} m_i = m$, $x_i \in R^n$ is the local variable, $z \in R^n$ is the global consensus variable, $f_i : R^n \to R$ is the cost function, and $g : R^n \to R \bigcap \{\infty\}$ is the regularization function.

In the formula (2), the objective function $f(x, D)$ is divided into N sub-problems $f_i(x_i, D_i)$, and the local variable x_i is required to be consistent with the global variable z, so it is very suitable for a parallel solution in the distributed environment. The distributed ADMM algorithm is implemented by MapReduce in a study by Lubell-Doughtie et al. [3] and by MPI in a study by Taylor et al. [4]. The distributed ADMM algorithms implemented in [3] and [4] are synchronous. Due to the difference in computing and communication performance between different nodes, the synchronization overhead becomes the bottleneck for shortening the running time of the algorithm, and the asynchronous distributed ADMM algorithm becomes a new research hotspot [5, 6, 7]. Compared with the synchronous ADMM algorithm, the asynchronous ADMM algorithm can better solve the "slow node" problem caused by network delay and the difference between nodes in the synchronous algorithm [7], as well as improve the convergence speed of the algorithm. However, the asynchronous ADMM algorithm often needs more iterations to make it converge. As the size of the distributed system increases, the algorithm makes convergence more difficult to achieve. Moreover, the increase in the number of nodes also makes the central node overloaded, affecting the overall performance of the system. However, with the advances of computer hardware technology, it is very common for a single machine to have multicore and multiprocessor in the modern high-performance distributed system. Message Passing Interface (MPI), which is one of the important means of parallel program development in multicore systems, has different communication mechanisms between the intra-nodes and inter-nodes: the processes in the same node transmit data through shared memory or cache, while processes in different nodes transmit data through the network interface [8, 9]. Moreover, the different communication mechanisms cause the unbalanced arrival problem [8].

In order to solve the problems such as the distributed ADMM algorithm has slow convergence speed, the center node is overloaded, the unbalanced arrival problem in the multicore cluster, and to improve the system communication efficiency and speed up algorithm convergence, we propose a group-based asynchronous distributed alternating direction method of multipliers (GAD-ADMM). In our proposal, every process is viewed as a unit, and all the workers are grouped according to the process allocation in the multicore cluster and the model similarity of datasets. One worker is selected as the group leader in each group. Each worker is responsible for the update of local variable and dual variable, and then sends the local variable to the group leader to update the group variable. Finally, the master collects the group variables to update the global variable. The master only communicates with the group leaders, and the worker communicates with the master indirectly through the group leader, thus reducing the communication between nodes and the load of the master. In order to solve the unbalanced arrival problem, the GAD-ADMM algorithm adopts an asynchronous protocol between groups.

The main contributions of this paper can be summarized as follows:

1. We propose a group-based asynchronous distributed ADMM, which improves the communication efficiency by reducing the communication between nodes in a multicore cluster, and accelerates the convergence of the algorithm by relaxing the constraint conditions on global consistency.
2. The asynchronous communication protocol is used between groups to further improve the convergence speed of the algorithm on the premise of ensuring the convergence of the algorithm.
3. Instead of directly transmitting the group local variable and dual variable, the group leader first performs related operations on the group local variable and dual variable before transmitting the result to the master, further reducing the communication between nodes and reducing the calculation load of the master.
4. The GAD-ADMM algorithm is implemented in a high performance multicore distributed cluster. The benchmark experiments show that the GAD-ADMM

algorithm can reduce the number of external iterations, improve communication efficiency, and reduce the total time cost of the system by 35 % compared with the AD-ADMM algorithm.

The remainder of the paper is organized as follows. The background and related work of ADMM are introduced in Section 2. Section 3 introduces the GAD-ADMM. The theoretical analysis of the GAD-ADMM is presented in Section 4. In Section 5, experiments on logistic regression (LR) solved by the GAD-ADMM are presented. Finally, we present the conclusion of this paper in Section 6.

2 BACKGROUND AND RELATED WORK

Many machine learning problems can be transformed into global consensus optimization problems. The distributed ADMM algorithm based on global consistency is used to solve the SVM problem by Zhang et al. [10] and solve the logistic regression problem by Lubell-Doughtie et al. [3]. The ADMM algorithm is introduced by Boyd et al. [2] to solve the global consensus optimization problem. The iterative formula is shown as follows:

$$x_i^{k+1} := \underset{x_i}{\operatorname{argmin}} \left(f_i(x_i, D_i) + \frac{\rho}{2} \left\| x_i + \frac{1}{\rho} y_i^k - z^k \right\|_2^2 \right), \tag{3}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2} \sum_{i=1}^{N} \left\| x_i^{k+1} + \frac{1}{\rho} y_i^k - z \right\|_2^2 \right), \tag{4}$$

$$y_i^{k+1} := y_i^k + \rho \left(x_i^{k+1} - z^{k+1} \right) \tag{5}$$

where $y_i \in R^N$ is the dual variable, ρ is the penalty parameter, and $\left\| x + \frac{1}{\rho} y - z \right\|$ is the penalty term. It is shown in Equations (3), (4) and (5) that the updates of local variables and dual variables can be executed parallel in different nodes, while the aggregation of all local variables and dual variables is desired to solve the global variable. Generally, one master can be used to update the global variable, and N workers can be used to update local variables and dual variables independently. Due to the fault tolerance of the machine learning algorithm and the decomposable characteristics of the algorithm, moderate relaxation of the accuracy requirements of the iterative process can make the algorithm converge faster.

According to different communication topologies, distributed ADMM algorithms can be classified into point-to-point mode [11] and master-slave mode [3, 12]. The global consensus optimization problems are usually solved using a master-slave mode. According to different communication protocols, distributed ADMM algorithms can be classified into synchronous and asynchronous distributed ADMM. In the synchronous distributed ADMM algorithm, the master must wait to receive the parameters of all workers before updating the global variable [2, 3, 4, 10], so the speed of the algorithm is limited by the slowest node. To solve this problem, Zhang et al. [7] proposed an asynchronous distributed ADMM (async-ADMM). A new asynchronous distributed ADMM (AD-ADMM) proposed by Chang et al. [12], which does not need the loss function, must be a convex function. Chang et al. [13] further proved that the AD-ADMM had linear convergence. Unlike the synchronous distributed ADMM algorithm, the AD-ADMM master only needs to receive the local variables from a partial list of workers to update the global variable z. As the number of workers increases, the convergence of the distributed ADMM algorithm becomes more difficult. To solve this problem, Wang et al. [14] proposed a group-based distributed ADMM (GADMM): all the workers are grouped into several groups by model similarity in the GADMM algorithm, and then the group variables are used to replace local variables and update the global variable. The convergence speed of the algorithm is improved by relaxing the constraint on global consistency.

The distributed ADMM algorithms introduced above optimize the iterative format and transmission process of the distributed ADMM algorithm from different angles, but do not consider the difference of data communication between intra-nodes and inter-nodes. This kind of communication variability often leads to the unbalanced arrival problem, which slows down the convergence speed of the algorithm. However, as the number of distributed system nodes increases, the distributed ADMM algorithm converges slowly. The GAD-ADMM algorithm proposed in this paper adds a layer of grouping on the basis of the AD-ADMM algorithm framework, where the workers are grouped according to the distribution of processes in the distributed system and the model similarity of datasets. In addition, it uses group variables instead of local variables to update the global variable. To improve the algorithm efficiency and solve the unbalanced arrival problem in the multicore cluster, a reasonable grouping of the workers was adopted for the GAD-ADMM. Besides, the convergence speed of the GAD-ADMM algorithm was accelerated by relaxing the constraints of global consistency. This paper bridges the structural characteristics of distributed systems with the characteristics of distributed algorithms and proposes an efficient grouping asynchronous distributed ADMM (GAD-ADMM) algorithm. The GAD-ADMM is also different from the GADMM in two ways:

1. the GAD-ADMM fully considers the communication differences between processes in a multicore cluster, and takes these differences as the main factor of process grouping;
2. instead of a synchronous protocol, the asynchronous protocol is used between groups.

3 THE GROUP-BASED ASYNCHRONOUS DISTRIBUTED ADMM (GAD-ADMM)

We define the notations included in the rest of this paper as follows.

Definition 1. P represents the number of nodes in the system and Q_i represents the number of working processes in node P_i. P_{ij} represents the j^{th} process in the i^{th} node. All the processes in the node P_i are divided into M_i groups, and M is the total number of groups. M_1 is the number of groups of the processes of the node where the master is located. N is the total number of workers. W_{ij} represents the j^{th} worker in the G_i^{th} group. D_{ij} represents the dataset processed by P_{ij} or W_{ij}, and C_{ij} is the number of samples of D_{ij}. The data transfer rate in the node is V_{in} B/s while between nodes is V_{out} B/s. The number of bytes occupied by the parameter x is F_{dim} (the number of bytes occupied by the parameter y or z is also F_{dim}).

As the size of the data increases, the algorithm becomes more and more difficult to converge. The main purpose of the GAD-ADMM is to improve communication efficiency and speed up the convergence of the algorithm by grouping the processes. Based on the traditional master-slave mode, the GAD-ADMM adds a grouping layer to form a two-layer master-slave mode. After grouping, there are three different types of processes in the system: the master, the group leader and the worker. The entire algorithm framework of GAD-ADMM consists of four steps. First, the processes of workers are classified into several groups according to the process allocation of the system and model similarity of datasets. Each group selects one process as the group leader. Second, the worker updates local variable and dual variable, then sends the local variable to the group leader, and finally waits to receive the group variable and global variable from the group leader for the next update. Third, the group leader gathers all the local variables of the group to produce the group local variable and dual variable, then the group leader must send the group variable to the master and wait to receive the global variable from the master, and finally broadcast the group local variable and global variable to the workers. Finally, the master gathers the group variables from group leaders to update the global variable and then sends the updated global variable to the group leaders. Due to the differences between groups and network delay, we adopt an asynchronous communication protocol between groups. The master does not wait for the parameters of all groups to arrive, but only waits for the parameters of some groups (the number of groups can be set by the user) to arrive to update the global variable. All these steps, with the exception of the first one, are iterated until the algorithm converges.

3.1 The Decomposition of the Calculation and the Grouping Method of the Process

In a multicore cluster, each node may be assigned multiple processes, and each process processes an independent dataset, and updates the local variable and dual variable in parallel. In the iterative process, formulae (3), (4) and (5) can be further modified as follows:

$$x_{ij}^{k+1} := \underset{x_{ij}}{\operatorname{argmin}} \left(f_i(x_{ij}, D_{ij}) + \frac{\rho}{2} \left\| x_{ij} + \frac{1}{\rho} y_{ij}^k - z^k \right\|_2^2 \right), \tag{6}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2} \sum_{i=1}^{P} \sum_{j=1}^{Q_i} \left\| x_{ij}^{k+1} + \frac{1}{\rho} y_{ij}^k - z \right\|_2^2 \right), \tag{7}$$

$$y_{ij}^{k+1} := y_{ij}^k + \rho \left(x_{ij}^{k+1} - z^{k+1} \right) \tag{8}$$

where D_{ij} represents the dataset processed by the process P_{ij}, and x_{ij} and y_{ij} represent the local variable and dual variable updated by P_{ij}, respectively. According to formula (7), the solution of the global variable needs to aggregate all local variables and dual variables. As the number of processes increases, the load on the master increases. This algorithm first performs group aggregation on local variables, and then uses the aggregate value of the group to update the global variable in order to reduce the load on the master. So how to divide the working processes into groups appropriately is considered in this phase. Considering the high cost of data communication between nodes, we first assign processes in the same node to the same group by default. Then, we analyze the impact of different groups on system time cost.

In each iteration, the master needs to aggregate the group variables of all groups, and then broadcast the global variable to each group. Therefore, the total system time (T_{total}) of the master includes the waiting time (T_{wait}), the calculation time (T_{cal}) and the sending time (T_{send}). The waiting time is determined by the update time of each group and the transfer time of model parameters. The calculation time is the time when the master calculates the global variable. The sending time is the time required for the master to transfer the global variable to the group leaders. The waiting time of each iteration can be expressed as follows:

$$T_{wait} = \overline{T_{update}} + T_{trans} \tag{9}$$

where $\overline{T_{update}}$ represents the average update time of all groups, and $\overline{T_{update}}$ is mainly determined by the update time of local variables in the workers, so the change of the number of groups does not have a great influence on it. T_{trans} is the transfer time, which can be computed as formula (10), and includes the time of the model parameters transferred between the master and the group leaders, and between the group leader and the workers:

$$T_{trans} = 3N \frac{F_{dim}}{V_{in}} + 3 \sum_{i=2}^{P} M_i \left(\frac{F_{dim}}{V_{out}} - \frac{F_{dim}}{V_{in}} \right). \tag{10}$$

The sending time can be computed as formula (11) and the calculation time can be computed as formula (12) in each iteration, in which T_{add} represents the time of an accumulation operation:

$$T_{send} = (M - M_1)\frac{F_{dim}}{V_{out}} + M_1\frac{F_{dim}}{V_{in}}, \tag{11}$$

$$T_{cal} = \sum_{i=1}^{M} T_{add}. \tag{12}$$

Usually, $V_{in} > V_{out}$. It can be induced from (9), (10), (11) and (12) that when the dataset and the number of processes are constant, as the number of groups M increases, T_{trans}, T_{cal} and T_{send} increase, and $\overline{T_{update}}$ is basically unchanged. So the total time of the system in one iteration increases. That is to say, in each iteration, the fewer the number of groups, the higher the system communication efficiency. So we first divide the processes in the same node into a group by default.

After grouping processes, the group variables are used instead of local variables to update the global variable to speed up the convergence of the algorithm. Therefore, we use the similarity of the dataset as another criterion for grouping, that is, the model similarity of the dataset of the processes in the same group after grouping is higher. To achieve this purpose, the processes in the same node are further grouped according to model similarity. In this algorithm, we use Euclidean distance as the measure of similarity: the smaller the Euclidean distance, the larger the similarity of datasets. Measuring model similarity of datasets also requires expensive computing and communication overhead, so we use the similarity of the local variable x to replace the model similarity of the dataset as the measurement indicator. The Euclidean distance (Ed) between the process P_{ij} and the process P_{ik} can be calculated by formula (13):

$$Ed(P_{ij}, P_{ik}) = \|x_{ij}^1 - x_{ik}^1\|^2 \tag{13}$$

where $x_{ij}^1, x_{ik}^1 \in R^n$ represent the first updated values of the local variables of P_{ij} and P_{ik}, respectively. After calculating the similarity between processes, the L algorithm [15] is used to determine the number of internal groups, and the DIANA algorithm [18] is used to perform regrouping operations on each default group. The DIANA algorithm is a top-down hierarchical clustering algorithm, which needs to determine the number of groups in advance, while the L algorithm can automatically determine the number of groups in the hierarchical clustering algorithm [15]. Other clustering algorithms can also be used to regroup the processes. Local variables are used instead of datasets as indicators for similarity measurement. Although it is necessary to perform an update operation on the local variables in advance, the updated values can be used as the initial value of subsequent iteration updates, so it will not increase the total system time cost.

3.2 Group-Based Parallel Iterative Update Strategy

The workers are responsible for updating local variables and dual variables, which can be executed in parallel. Each worker first updates local variable and dual variable, then sends the local variable to the group leader, waits to receive the group variable and global variable from the group leader, and then updates the local variable again with the group variable. x_{ij} represents the local variable and y_{ij} represents dual variable of the j^{th} worker in the group G_i. Using formulae (14) and (15), each worker updates local variable and dual variable, respectively,

$$x_{ij}^{k_i+1} := \underset{x_{ij}}{\operatorname{argmin}}(f_i\left(x_{ij}, D_{ij}) + \frac{\rho}{2}\left\|x_{ij} + \frac{1}{\rho}y_{ij}^{k_i} - \widetilde{z}_{G_i}\right\|_2^2\right), \tag{14}$$

$$y_{ij}^{k_i+1} := y_{ij}^{k_i} + \rho\left(x_{ij}^{k_i+1} - \widetilde{z}_{G_i}\right) \tag{15}$$

where $D_{ij} \in R^{m_{ij}*n}$, $x_{ij} \in R^n$, $y_{ij} \in R^n$, $\sum_{i=1}^{P}\sum_{j=1}^{Q_i} m_{ij} = m$, and $\widetilde{z}_{G_i}$ represents the latest z-value received by the group G_i. In the GAD-ADMM, the workers in the same group are synchronous, while in different groups they are asynchronous, so the workers in the same group have the same $\widetilde{z}_{G_i}$ while workers in different group may have different $\widetilde{z}_{G_i}$. The procedure for the worker is described in Algorithm 1.

Because the datasets are grouped according to model similarity, the local variable is set to the same as the group local variable. It is worth noting that since the local variable is updated at the time of grouping, the first update of the local variable is directly set equal to the initial value, which is the updated value of the local variable in the process grouping. Update of the local variable is the optimal value for solving the sub-problem (14). In this paper, we use the Trust Region Newton method (TRON) [16] to solve the sub-problem. Of course, other optimization methods are also applicable to this algorithm.

Algorithm 1: Group-based Asynchronous Distributed ADMM (GAD-ADMM): Processing by worker j in the group G_i

```
Initialize $x_{ij}^0$, $y_{ij}^0$ and set $k_i$=0.
set $x_{ij}^1 = x_{ij}^0$ and update $y_{ij}^{k_i+1}$ using (15)
send $x_{ij}^1$ to the group leader
repeat
    wait until receiving $\widetilde{z}_{G_i}$ and $x_{G_i}$ from the group leader.
    set $k_i = k_i + 1$
                        $x_{ij} = x_{G_i}$                    (16)
    update $x_{ij}^{k_i+1}, y_{ij}^{k_i+1}$ using (14), (15)
    send $x_{ij}^{k_i+1}$ to the group leader.
until the stopping conditions are satisfied;
```

3.3 Group-Based Parameter Aggregation Communication

After grouping the processes, the process with the minimum process number is selected as the group leader of the group. The group leader gathers all local variables from the workers in the group, then updates the group variable x_{G_i} and the group dual variable y_{G_i}. After the group leader updates the values of $x_{G_i}^{k_i+1}$ and $y_{G_i}^{k_i+1}$, the updated values can be directly sent to master. It can be seen from Equation (7) that, when solving the global variable, we need to sum up $y_i/\rho + x_i$. Therefore, the calculation can be performed prior to the transmission, leading the calculation result w_{G_i} (w_{G_i} should be computed as formula (17)) transferred to the master directly instead of x_{G_i} and y_{G_i}. In this case, the transfer time can be calculated by formula (18), it can be induced by formulae (10) and (18) that $T'_{trans} < T_{trans}$:

$$w_{G_i}^{k_i+1} = y_{G_i}^{k_i+1}/\rho + x_{G_i}^{k_i+1}, \tag{17}$$

$$T'_{trans} = 3N\frac{F_{dim}}{V_{in}} + 2\sum_{i=2}^{P} M_i\left(\frac{F_{dim}}{V_{out}} - \frac{F_{dim}}{V_{in}}\right). \tag{18}$$

According to the grouping method, the model similarity of a dataset between workers in the same group is relatively high, and workers in the same group are in the same node, so the synchronous communication protocol is adopted between workers in the group. Considering that the number of samples on each worker may be different, the group variable is calculated from the weighted average of local variables of all the workers in each group. We define η_{ij} as the ratio of dataset D_{ij} in the group G_i, and η_{ij} can be computed by formula (19). Then, the group variable x_{G_i} can be updated as shown in formula (20), and the group dual variable y_{G_i} can be updated using formula (21):

$$\eta_{ij} = C_{ij}/\sum_{W_{ij}\in G_i} C_{ij}, \tag{19}$$

$$x_{G_i}^{k_i+1} := \sum_{W_{ij}\in G_i} \eta_{ij} x_{ij}^{k_i+1}, \tag{20}$$

$$y_{G_i}^{k_i+1} := y_{G_i}^{k_i} + \rho\left(x_{G_i}^{k_i+1} - \tilde{z}_{G_i}\right) \tag{21}$$

where $\tilde{z}_{G_i}$ represents the latest z-value received by the group leader of group G_i. After this, the group leader sends w_{G_i} to the master, then waits to receive the global variable from the master, and finally broadcasts the group local variable and global variable to the workers. The whole procedure for the group leader is shown in Algorithm 2.

Algorithm 2: Group-based Asynchronous Distributed ADMM (GAD-ADMM): Processing by the group leader of the group G_i

```
Initialize $x_{G_i}, y_{G_i}, z_{G_i}$ and set $k_i$=0.
repeat
    wait $x_{ij}$ from all workers in the group $G_i$
    update $\{x_{G_i}^{k_i+1}, y_{G_i}^{k_i+1}, w_{G_i}^{k_i+1}\}$ using (20), (21), (17)
    send $\{w_{G_i}^{k_i+1}\}$ to the master
    wait until receiving $\widetilde{z}_{G_i}$from the master.
    broadcast $\widetilde{z}_{G_i}$ and $x_{G_i}$ to all workers in the group $G_i$.
    set $k_i = k_i + 1$
until the stopping conditions are satisfied;
```

3.4 Global Variable Data Synchronization and Update Based on Bounded Delay

The master waits to receive the group variables from the group leaders, updates the global variable z with the group variables, and then sends the new global variable to the corresponding group leaders. Only a partial synchronization is required for each iteration of the master, not full synchronization for all groups. The master only broadcasts the global variable to the corresponding groups in each iteration. However, since the number of processes in each group may be different, we use the weighted value of the grouped variables to solve the global variable. Assuming that there are N_{G_i} processes in the group G_i, the global variable can be updated using formulae (22) and (23).

$$w_{G_i}^{k+1} = \begin{cases} \widetilde{w}_{G_i}, & \forall G_i \in A_k, \\ w_{G_i}^{k}, & \forall G_i \in A_k^c, \end{cases} \tag{22}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2} \sum_{i=1}^{M} \frac{MN_{G_i}}{N} \left\| w_{G_i}^{k+1} - z \right\|_2^2 + \frac{\theta}{2} \left\| z - z^k \right\|_2^2 \right) \tag{23}$$

where A_k represents the index subset of group leaders received by the master in iteration k, A_k^c represents the complementary set of A_k, and ρ and θ are the penalty parameters. The penalty term $\left\| z - z^k \right\|_2^2$ is added to further ensure the convergence of the algorithm. The whole procedure for the master is shown in Algorithm 3.

In the GAD-ADMM, the bulk synchronous parallel (BSP) mode is adopted between workers in the same group, while the stale synchronous parallel (SSP) [17] mode is adopted between groups. The workers in the same group have the same iteration cycles and between groups may have different iteration cycles. The SSP mode achievement involves several steps. First, the minimum synchronized block size is set to A ($M \geq A \geq 1$), indicating that the global variable z will not be updated until the master has successfully received the group variables from A group

Algorithm 3: Group-based Asynchronous Distributed ADMM (GAD-ADMM): Processing by the master

```
Initialize $z$ and set $k = 0$, $d_{G_i} = \ldots = d_{G_M} = 0$.
repeat
    wait until receiving a minimum of A updated variables from the group
      leaders and $d_{G_i} \leq d$ for all $i \in \{1, \ldots, M\}$
    update
        $d_{G_i} = \begin{cases} 0 : \forall G_i \in A_k \\ d_{G_i+1} : \forall G_i \in A_k^c \end{cases}$        (24)
      update $z^{k+1}$ using (23).
    broadcast $z^{k+1}$ to the group leaders in $A_k$.
    set $k = k + 1$
until the stopping conditions are satisfied;
```

leaders. Second, the maximum delay cycle is set to d ($d \geq 1$), and every group must update at least once in this cycle. Every group leader has its own delay counter d_{G_i}, which is stored on the master. When the w_{G_i} from the group G_i arrives at the master, the corresponding d_{G_i} is set to 0, otherwise, d_{G_i} is increased by one as the master's counter k increments. The d_{G_i} for each group must be less than d. The GAD-ADMM reduces to synchronous when A is equal to M or d is set to 1. Figure 1 shows the timing diagram of the GAD-ADMM when the threshold A is set to 2 and cycle d is set to 3.

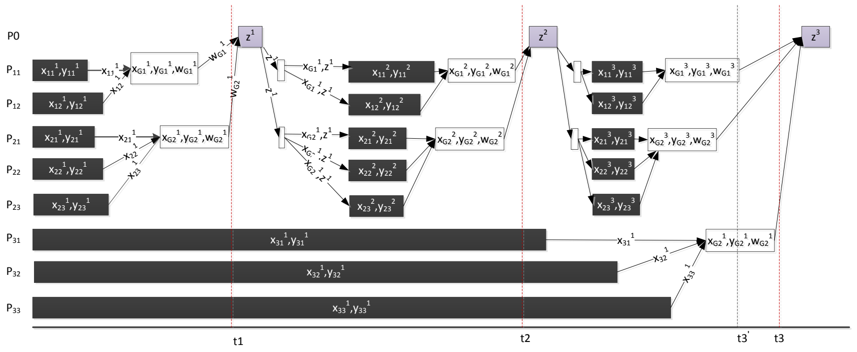


Figure 1. The timing diagram of the GAD-ADMM ($A = 2$, $d = 3$): P_0 represents the master, and P_{ij} represents the j^{th} worker in the group G_i. In this figure, there are 8 workers grouped into 3 groups, in which P_{11}, P_{21} and P_{31} are the group leaders of the corresponding groups. The number in the upper right corner of the variable represents the number of iterations.

4 THEORETICAL ANALYSIS

In this section, we analyze the convergence of the GAD-ADMM and further compare the convergence speed of the GAD-ADMM with the AD-ADMM.

4.1 Convergence Analysis

In this section, we analyze the convergence of the GAD-ADMM, in which the augmented Lagrangian for (2) can be modified to the following formula:

$$L_\rho(x,y,z) = \sum_{i=1}^{M} f_i(x_{G_i}) + g(z) + \sum_{i=1}^{M} \frac{MN_{G_i}}{N} \left(\langle y_{G_i}, x_{G_i} - z \rangle + \frac{\rho}{2} \|x_{G_i} - z\|_2^2 \right). \quad (25)$$

First, we make the assumption as follows to simplify analysis.

Assumption 1. The function g is proper, closed and convex; Each function f_i is a convex function, and there is a constant L ≥ 0 such that the gradient of f_i satisfies the Lipschitz continuous condition; Moreover, problem (2) has an optimal solution $f^\star$, which is bounded; Set $d \geq 1$ as the maximum bounded delay; For all $i \in \{1, \ldots, M\}$ and $k \geq 0$, it must be that $i \in A_k \bigcup A_{k-1} \bigcup \ldots \bigcup A_{max(k-d+1,-1)}$, and there exists a constant $B \in [1, M]$ such that $|A_k| \leq B$ for all k ($|A_k|$ represents the number of A_k).

Theorem 1. If Assumption 1 is true, then the sequence of $\left(\{x_i^k\}_{i=1}^N, \{y_i^k\}_{i=1}^N, z^k\right)$ generated by the GAD-ADMM is bounded and has limit points, which satisfy the KKT conditions of problem (2) under the condition that formulae (26), (27), (28) are established:

$$\infty > L_\rho\left(x^0, z^0, y^0\right) - f^\star \geq 0, \quad (26)$$

$$\rho \geq \left(\left(1 + L^2\right) + \sqrt{\left(1 + L^2\right)^2 + 8L^2} \right) / 2, \quad (27)$$

$$\theta > \left(B\left(1 + \rho^2\right)(d-1)^2 - M\rho \right) / 2. \quad (28)$$

The proof of Theorem 1 is similar to that of Corollary 1 in [12]. It is implied by Theorem 1 that the GAD-ADMM is guaranteed to converge to the set of KKT points so long as ρ and θ are large enough. The reciprocal of θ can be considered as the step size of z.

4.2 Convergence Speed Analysis

We further analyze the convergence speed of the GAD-ADMM and the AD-ADMM in this section. First, we analyze the update speed of the local variable for each worker. We define the objective function of the sub-problem as follows:

$$F(x_i) = f(x_i) + h(x_i) \quad (29)$$

where $h(x_i) = \frac{\rho}{2}\|x_i - z + \frac{1}{\rho}y_i\|^2$, $f(x_i)$ is the loss function and $f(x_i)$ is convex. We also denote $x_i^{k_i}$ as the optimal value of the worker i in iteration k_i.

Lemma 1. Assuming that the AD-ADMM and the GAD-ADMM have the same method for updating the global variable z, all N workers in the GAD-ADMM are classified into M groups. For any $k_i \geq 1$, the local variable $x_i^{k_i}$ sequence satisfies the formula (30):

$$\sum_{i=1}^{N} F(x_i^{k_i}) \geq \sum_{i=1}^{M} N_i F\left(\sum_{W_{ij}\in G_i} \eta_{i,j} x_{ij}^{k_i}\right) \tag{30}$$

where N_i represents the number of the workers in the group G_i, and $\eta_{i,j}$ is the weight of W_{ij} in the group G_i.

Proof. We first analyze the situation that there are only two workers (W_a and W_b) in the group, and we define x_c is the average of the local variables x_a and x_b. In the GAD-ADMM, the workers in the same group are synchronous, so the workers in the same group have the same k_i and $\widetilde{z}_{G_i}$. Because $f(x)$ is convex, we can induce that

$$f(x_a) + f(x_b) \geq 2f(x_c) \tag{31}$$

when $k_i = 2$, it can be deduced from (14), (15) and (31) that

$$\begin{aligned} F(x_a^2) + f(x_b^2) - 2f(x_c^2) &\geq (\rho/2)\left\|x_a^2 - \widetilde{z}_{G_i}^1 + (1/\rho)y_a^1\right\|^2 \\ &+ (\rho/2)\left\|x_b^2 - \widetilde{z}_{G_i}^1 + (1/\rho)y_b^1\right\|^2 - \rho\left\|x_c^2 - \widetilde{z}_{G_i}^1 + (1/\rho)y_c^1\right\|^2 \geq (\rho/2)\left\|x_a^2 + x_a^1 - \widetilde{z}_{G_i}^1\right\|^2 \\ &+ (\rho/2)\left\|x_b^2 + x_b^1 - \widetilde{z}_{G_i}^1\right\|^2 - (\rho/4)\left\|x_a^2 + x_a^1 + x_b^2 + x_b^1 - 2\widetilde{z}_{G_i}^1\right\|^2. \end{aligned} \tag{32}$$

Similar, when $k_i > 2$,

$$\begin{aligned} F(x_a^{k_i}) + f\left(x_b^{k_i}\right) - 2f\left(x_c^{k_i}\right) \geq (\rho/2)&\left\|\sum_{i=1}^{k^i} x_a^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \\ &+ (\rho/2)\left\|\sum_{i=1}^{k^i} x_b^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 - \rho\left\|\sum_{i=1}^{k^i} x_c^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \\ &\geq (\rho/2)\left\|\sum_{i=1}^{k^i} x_a^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 + (\rho/2)\left\|\sum_{i=1}^{k^i} x_b^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \\ &- (\rho/4)\left\|\sum_{i=1}^{k^i} x_b^i + \sum_{i=1}^{k^i} x_b^i - 2\sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \geq 0. \end{aligned} \tag{33}$$

If there are p workers in a group, then in the k_i^{th} iteration, it can be deduced that

$$\sum_{i=1}^{p} F\left(x_i^{k_i}\right) - pF\left(\frac{1}{p}\sum_{i=1}^{p} x_i^{k_i}\right) \geq (\rho/2)\sum_{j=1}^{p}\left\|\sum_{i=1}^{k_i} x_j^i - \sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2 - \frac{\rho}{2p}\left\|\sum_{i=1}^{k_i}\sum_{j=1}^{p} x_j^i - p\sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2 \geq 0. \tag{34}$$

When all workers are divided into M groups, and the number of workers in the G_i group is N_i, the difference between the original function $F(x_i)$ and the function $F(x_i)$ after grouping is:

$$\sum_{i=1}^{N} F\left(x_i^{k_i}\right) - \sum_{i=1}^{M} N_i F\left(\sum_{W_{ij}\in G_i}\eta_{i,j}x_{ij}^{k_i}\right) = \sum_{i=1}^{N} F\left(x_i^{k_i}\right) - \sum_{i=1}^{M} N_i F\left(\frac{1}{N_i}\sum_{j=1}^{N_i} x_{ij}^{k_i}\right)$$
$$\geq \sum_{i=1}^{M}\left\{\sum_{j=1}^{N_i}\left\|\sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2 - \frac{\rho}{2N_i}\left\|\sum_{i=1}^{k_i}\sum_{j=1}^{N_i} x_{ij}^i - N_i\sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2\right\} \geq 0. \tag{35}$$

□

The main difference between the GAD-ADMM and the AD-ADMM is based on the sum of the local variable $x_i^{k_i}$ sequence, so $F(\eta x_i)$ is going down faster than $F(x_i)$.

Moreover, we define the objective function of the AD-ADMM and the GAD-ADMM as follows:

$$G(x^k, z^k) = \sum_{i=1}^{N} f_i(x_i^k) + g(z^k), \tag{36}$$

$$\bar{G}\left(\bar{x}^k, \bar{z}^k\right) = \sum_{i=1}^{N} f_i\left(\bar{x}_i^k\right) + g\left(\bar{z}^k\right). \tag{37}$$

Theorem 2. If $g(z) = \beta\|z\|^2$and the synchronous communication protocol is used both in the GAD-ADMM and the AD-ADMM, then

$$G(x^k, z^k) - \bar{G}(\bar{x}^k, \bar{z}^k) \geq \frac{\rho^2 M(N-M)}{N^2(2\beta+M\rho)^2(2\beta+N\rho)}\left\|v_i^{k+1}\right\|^2 \tag{38}$$

where $v_i^{k+1} = \sum_{i=1}^{N}\left(x_i^{k+1} + \frac{1}{\rho}y_i^{k+1}\right)$.

Proof. Since $g(z) = \beta\|z\|^2$, then it can be induced by (16),(20) and (23) that

$$\bar{x_i}^k = \sum_{W_{ij}\in G_i} \eta_{i,j} x_{ij}^k, \tag{39}$$

$$\bar{z}^k = \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{M} N_i \left(x_{G_i}^k + (1/\rho) y_{G_i}^k\right). \tag{40}$$

Since $f(x)$ is a convex function and non-negative, when the N workers are grouped into M groups, the loss function of sub-problem satisfies:

$$\sum_{i=1}^{N} f_i(x_i^k) \geq \sum_{i=1}^{M} N_i f_i \left(\frac{1}{N_i} \sum_{W_{ij}\in G_i} x_{ij}^k\right) \geq \sum_{i=1}^{M} N_i f_i \left(\sum_{W_{ij}\in G_i} \eta_{i,j} x_{ij}^k\right). \tag{41}$$

It can be induced that

$$z^{k+1} = \frac{\rho}{2\beta + N\rho} \sum_{i=1}^{N} \left(x_i^{k+1} + (1/\rho) y_i^{k+1}\right), \tag{42}$$

$$\begin{aligned} \bar{z}^k &= \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{M} N_i \left(x_{G_i}^k + (1/\rho) y_{G_i}^k\right) \\ &\leq \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{M} \sum_{W_{ij}\in G_i} \left(x_{ij}^{k+1} + (1/\rho) y_{ij}^{k+1}\right) \\ &\leq \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{N} \left(x_i^{k+1} + (1/\rho) y_i^{k+1}\right) \leq z^{k+1}, \end{aligned} \tag{43}$$

$$\begin{aligned} g\left(z^{k+1}\right) - g\left(\bar{z}^{k+1}\right) &\geq g\left(\frac{\rho v_i^{k+1}}{2\beta + N\rho}\right) - g\left(\frac{M\rho v_i^{k+1}}{N(2\beta + M\rho)}\right) \\ &\geq \frac{M(\rho v_i^{k+1})^T}{N(2\beta + M\rho)} \left(\frac{\rho v_i^{k+1}}{2\beta + N\rho} - \frac{M\rho v_i^{k+1}}{N(2\beta + M\rho)}\right) \\ &= \frac{\rho^2 M(N-M)}{N^2(2\beta + M\rho)^2(2\beta + N\rho)} \left\|v_i^{k+1}\right\|^2. \end{aligned} \tag{44}$$

Therefore, it can be induced by (41) and (44) that

$$G\left(x^k, z^k\right) - \bar{G}\left(\bar{x}^k, \bar{z}^k\right) \geq g\left(z^{k+1}\right) - g\left(\bar{z}^{k+1}\right) \geq \frac{\rho^2 M(N-M)}{N^2(2\beta + M\rho)^2(2\beta + N\rho)} \left\|v_i^{k+1}\right\|^2. \tag{45}$$

□

Theorem 2 indicates that when $M < N$, the difference between the object function of the AD-ADMM and the GAD-ADMM is greater than 0, and the smaller the M, the greater the difference. It also indicates that when M is equal to N, the GAD-ADMM is equal to the AD-ADMM.

5 EXPERIMENTS AND DISCUSSION

In this section, we solve the logistic regression problem with L2 regularization by the GAD-ADMM and the AD-ADMM [12]. The problem is described as follows:

$$\min \frac{1}{N}\sum_{i=1}^{N} l_i\left(D_i^T x_i - b_i\right) + \beta\|z\|^2, \quad \text{s.t. } x_i = z, i = 1, 2, \ldots, N. \tag{46}$$

where $D \in R^{m*n}$ is the sample dataset, m represents the number of samples, n represents the number of features of samples, $b_i \in \{0, 1\}$ is the label of the sample, and $\beta > 0$ is the scalar regularization parameter.

5.1 Experimental Environment and Parallel Implementation

The experimental platform of this paper is the Ziqiang 4000 high-performance cluster of Shanghai University. Each node of the cluster has an Intel E5-2690 CPU (2.9 GHz/8-core) processor and 64 GB memory (RDIMM DDR3 1 600 MHz), the network is gigabit Ethernet. We use the kdd2010 (bridge to algebra)[1] as the datasets, which contains 19 264 097 training samples and 1 163 024 features. All datasets are divided into training/testing datasets at a ratio of 3:1 for experimental testing. We implement the algorithm using the MPI implementation MPICH v3.2.1[2] as the inter-processor communication and C++ as the programming language.

The parallel implementation of GAD-ADMM and AD-ADMM uses eight compute nodes, and each node in the system is allocated with eight processes. We select one process as the master and other processes as the workers. The scalar regularization parameter β is set to 2. In the GAD-ADMM algorithm, we use the L algorithm to obtain the total number of groups M, which is eight.

The TRON [16] method is used to solve the local variables. The parameter ρ is set to 6 and the super parameter C is set to 1. We use the primal residual r and dual residual s as the stop conditions, which should satisfy the conditions shown in (47) and (48) to stop the iteration.

$$\|r^k\|_2 \le \text{ABS} * \sqrt{Mn} + \text{REL} * \max\left\{\|x_{G_i}^k\|_2, \|z^k\|_2\right\}, \tag{47}$$

$$\|s^k\|_2 \le \text{ABS} * \sqrt{Mn} + \text{REL} * \left\|\rho y_{G_i}^k\right\|_2 \tag{48}$$

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

[2] `http://www.mpich.org/`

where $\left\|r^k\right\|_2^2 = \sum_{i=1}^{M} \left\|x_{G_i}^k - z^k\right\|_2^2$, $\left\|s^k\right\|_2^2 = M\rho^2 \left\|z^k - z^{k-1}\right\|_2^2$. The absolute error ABS and relative error REL are both set to 0.001.

5.2 Experiment Test and Analysis

In this section, the performance of the GAD-ADMM is compared with the AD-ADMM in terms of the convergence speed, the system cost and the accuracy. Furthermore, the influences caused by the different parameters and different groups are also analyzed for the GAD-ADMM.

5.2.1 Convergence and Convergence Speed Test

In this section, the convergence of the GAD-ADMM and the AD-ADMM will be tested. The convergence of these two algorithms is compared when different values of threshold A are taken. We set $d = 5$, $\theta = 0$ in both of the algorithms. The results are shown in Figures 2 and 3.

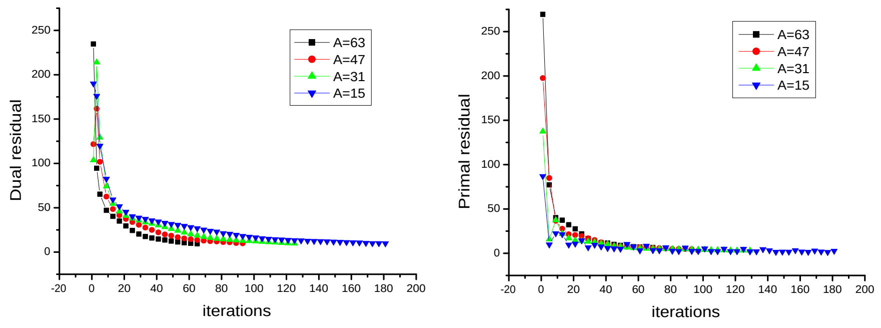


Figure 2. The convergence of the AD-ADMM

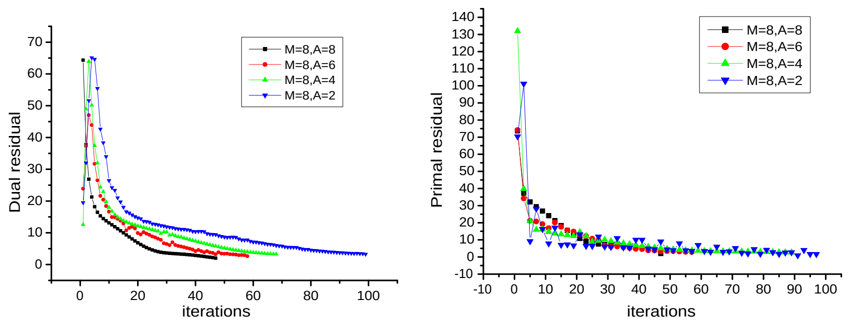


Figure 3. The convergence of the GAD-ADMM

Figures 2 and 3 show that the GAD-ADMM algorithm requires fewer iterations than that of the AD-ADMM to converge in a peer-to-peer situation. This is because in the GAD-ADMM, we relax the constraint conditions on global consensus, and the objective function is faster to converge as analyzed in Section 4. When M is fixed, the bigger the value of threshold A, the fewer the iterations required to converge. Because the more information the master receives from the leaders in each iteration, the less the "old values" used to update z, and the faster the algorithm converges. When A is larger, it may take longer for the master to wait for receiving, which shown in the next section.

5.2.2 System Time Cost and Accuracy Test

This section tests the system time cost and accuracy of the AD-ADMM and the GAD-ADMM. The system time cost includes the waiting time, the calculation time and the sending time. The accuracy of the system (Accur) is computed as follows:

$$\text{Accur} = (N_{tp} + N_{tz})/N_{total} \tag{49}$$

where N_{tp} represents the number of "true positive", N_{tz} represents the number of "true zero" and N_{total} represents the total number of testing datasets. The parameters of this section are set in accordance with Section 5.2.1. Figure 4 shows the system time cost of the master of the AD-ADMM and the GAD-ADMM in different thresholds. Figure 5 shows the accuracy of these two algorithms.

Figure 4 shows that the GAD-ADMM algorithm requires less system time than that of the AD-ADMM in a peer-to-peer situation. This is because only the group leader communicates directly with the master in the GAD-ADMM, which reduces the traffic between nodes. Thus, the waiting time and sending time for each iteration are reduced. At the same time, according to Equation (12), the smaller the number of groups, the less time each calculation takes. Furthermore, the smaller the number of groups, the less the number of external iterations, so the total system time cost is reduced. If M is constant, when the threshold value A decreases, the total number of iterations increases, and the calculation time increases, but the waiting time and sending time for each iteration decreases. So selecting appropriate A can minimize the system total time.

Figure 5 shows that compared with the AD-ADMM, the accuracy of the GAD-ADMM decreases, that is because the convergence conditions are relaxed. It also shows that the accuracy decreases less than 0.2 % when the total time is reduced by at least 35 % under peer conditions.

5.2.3 System Time Cost Test with Different Parameters

In Section 3.3, we mentioned that the leader can send parameters x and y to the master respectively, or send parameter w to the master. Figure 6 shows the system time allocation of the GAD-ADMM and the AD-ADMM in these two cases when the threshold A is equal to the group number M.

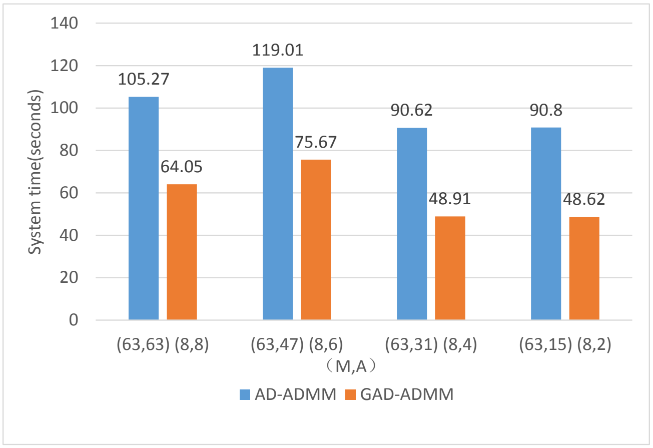


Figure 4. The system time cost of the AD-ADMM and the GAD-ADMM: A is the threshold and M is the number of groups

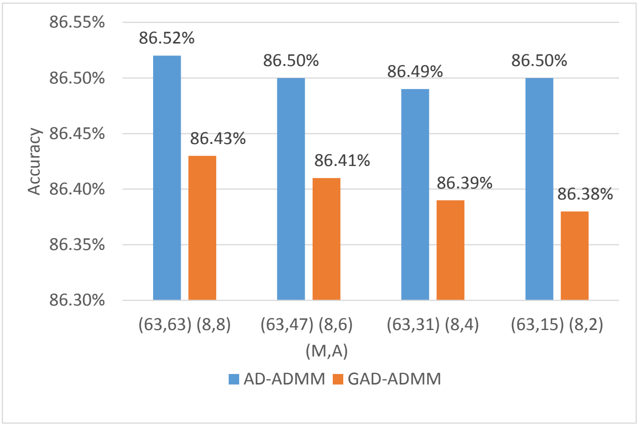


Figure 5. The accuracy of the AD-ADMM and the GAD-ADMM: A is the threshold and M is the number of groups

Figure 6 shows that when the group leader sends w to the master, the waiting time is reduced compared to sending x and y. This is because the parameters that the group leader transmits to the master each time are reduced by nearly half. Compared with the AD-ADMM, the GAD-ADMM does not have obvious advantages when transferring w. That is because the number of processes communicating directly with the master decreases after grouping, and the performance improvement is not obvious when the system bandwidth is high.

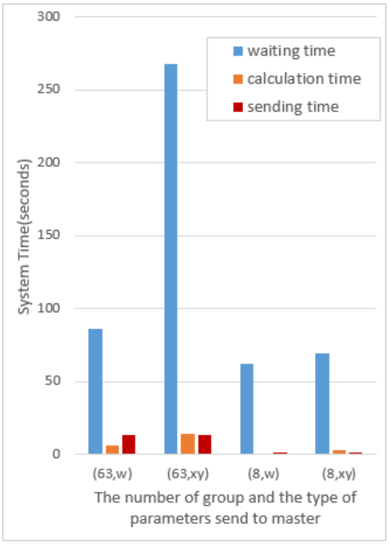


Figure 6. The system time allocation of the AD-ADMM and the GAD-ADMM when sending different types of parameters

5.2.4 System Time Cost Test of GAD-ADMM with Different Groups

In order to analyze the effects of different groups on the system time cost of the GAD-ADMM, the processes within the nodes are divided into groups 1, 2, 4 and 8 (the number of groups of the node where the master is located is 7), that is, M is taken as 8, 16, 32, and 63. Figure 7 shows the allocation of system time cost of the master under different groups and different thresholds. It shows that the system time cost decreases as the number of groups decreases in peer to peer conditions. The main reason is that the reduction of the number of groups makes the system converge faster. Another reason is that the system communication efficiency is improved by reducing the communication between nodes.

6 CONCLUSION

The GAD-GADMM algorithm is proposed in this paper for global consensus optimization problem. A grouping layer is added to the star topology, and the processes are grouped according to the process allocation in the multicore cluster and the model similarity of datasets. The convergence speed of the algorithm is improved by relaxing the global consistency constraint and reducing the communication between nodes. In the GAD-ADMM, the workers in the same group are synchronous while in different groups they are asynchronous, and we use partial barrier and bounded delay to guarantee the convergence of the asynchronous algorithm. In order to re-

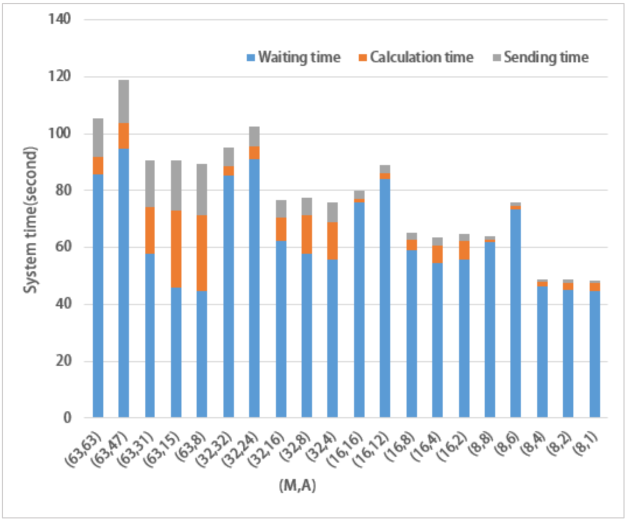


Figure 7. The system time allocation of the GAD-ADMM with different groups (M) and thresholds (A)

duce the master load, the parameter that the group leaders send to the master is w, not x and y, thus further reducing the system time cost. The convergence of the GAD-ADMM is proved both in theory and experiments. We also explained why the GAD-ADMM has a faster convergence speed than the AD-ADMM. The experiments on LR problem show that compared with the AD-ADMM, the GAD-ADMM can reduce the total system time by at least 35 % when the accuracy is reduced less than 0.2 % under peer conditions. Finally, the effects of different parameters and different number of groups on the performance of the GAD-ADMM algorithm were analyzed. The grouping method proposed in this paper first divides the processes in the same node into a group by default, and then further groups the default groups. The minimum number of groups is the number of nodes. It is not considered that the number of groups is less than the number of nodes, which will be studied in the future.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under grant No. U1811461.

REFERENCES

[1] Xing, E. P.—Ho, Q.—Xie, P.—Wei, D.: Strategies and Principles of Distributed Machine Learning on Big Data. Engineering, Vol. 2, 2016, No. 2, pp. 179–195, doi: 10.1016/j.eng.2016.02.008.

[2] Boyd, S.—Parikh, N.—Chu, E.—Peleato, B.—Eckstein, J.: Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Foundations and Trends in Machine Learning, Vol. 3, 2011, No. 1, pp. 1–122, doi: 10.1561/2200000016.

[3] Lubell-Doughtie, P.—Sondag, J.: Practical Distributed Classification Using the Alternating Direction Method of Multipliers Algorithm. IEEE International Conferences on Big Data, Vol. 1, 2013, No. 1, pp. 773–776, doi: 10.1109/bigdata.2013.6691651.

[4] Taylor, G.—Burmeister, R.—Xu, Z.—Singh, B.—Patel, A.—Goldstein, T.: Training Neural Networks Without Gradients: A Scalable ADMM Approach. Proceedings of the 33rd International Conference on Machine Learning (ICML 2016). The Journal of Machine Learning Research: Workshop and Conference Proceedings, Vol. 48, 2016, pp. 2722–2731.

[5] Wei, E.—Ozdaglar, A.: On the $O(1/k)$ Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers. IEEE Global Conference on Signal and Information Processing, 2013, pp. 551–554, doi: 10.1109/globalsip.2013.6736937.

[6] Shi, W.—Ling, Q.—Yuan, K.—Wu, G.—Yin, W.: On the Linear Convergence of the ADMM in Decentralized Consensus Optimization. IEEE Transactions on Signal Processing, Vol. 62, 2014, No. 7, pp. 1750–1761, doi: 10.1109/tsp.2014.2304432.

[7] Zhang, R.—Kwok, J. T.: Asynchronous Distributed ADMM for Consensus Optimization. Proceedings of the 31th International Conference on Machine Learning (ICML 2014). The Journal of Machine Learning Research: Workshop and Conference Proceedings, Vol. 32, 2014, pp. II-1701-II-1709.

[8] Patarasuk, P.—Yuan, X.: Efficient MPI Bcast across Different Process Arrival Patterns. IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–11, doi: 10.1109/ipdps.2008.4536308.

[9] Tipparaju, V.—Nieplocha, J.—Panda, D.: Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters. Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS '03), 2003, Art. No. 84, doi: 10.1109/ipdps.2003.1213188.

[10] Zhang, C.—Lee, H.—Shin, K. G.: Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers. Proceedings of the 15th International Conference on Artificial Intelligence and Statistics, La Palma, Spain, 2012, pp. 1398–1406.

[11] Mota, J. F. C.—Xavier, J. M. F.—Aguiar, P. M. Q.—Püschel, M.: D-ADMM: A Communication-Efficient Distributed Algorithm for Separable Optimization. IEEE Transactions on Signal Processing, Vol. 61, 2013, No. 10, pp. 2718–2723, doi: 10.1109/tsp.2013.2254478.

[12] Chang, T. H.—Hong, M.—Liao, W. C.—Wang, X.: Asynchronous Distributed ADMM for Large-Scale Optimization – Part I: Algorithm and Convergence Analysis. IEEE Transactions on Signal Processing, Vol. 64, 2016, No. 12, pp. 3118–3130, doi: 10.1109/TSP.2016.2537271.

[13] Chang, T. H.—Hong, M.—Liao, W. C.—Wang, X.: Asynchronous Distributed ADMM for Large-Scale Optimization – Part II: Linear Convergence Analysis and Numerical Performance. IEEE Transactions on Signal Processing, Vol. 64, 2016, No. 12, pp. 3131–3144, doi: 10.1109/TSP.2016.2537261.

[14] Wang, H.—Gao, Y.—Shi, Y.—Wang, R.: Group-Based Alternating Direction Method of Multipliers for Distributed Linear Classification. IEEE Transactions on Cybernetics, Vol. 47, 2017, No. 11, pp. 3568–3582, doi: 10.1109/tcyb.2016.2570808.

[15] Salvador, S.—Chan, P.: Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms. Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2004, pp. 576–584, doi: 10.1109/ictai.2004.50.

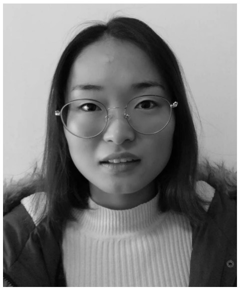
[16] Lin, C. J.—Weng, R. C.—Keerthi, S. S.: Trust Region Newton Methods for Large-Scale Logistic Regression. Proceedings of the 24th International Conference on Machine Learning (ICML 2007), Corvalis, Oregon, USA, 2007, pp. 561–568, doi: 10.1145/1273496.1273567.

[17] Ho, Q.—Cipar, J.—Cui, H.—Kim, J. K.—Lee, S.—Gibbons, P. B.—Gibson, G. A.—Ganger, G. R.—Xing, E. P.: More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS '13), Vol. 1, 2013, pp. 1123–1231.

[18] Kaufman, L.—Rousseeuw, P. J.: Finding Groups in Data: An Introduction to Cluster Analysis. New York, NY, USA, Wiley, 1990.

Dongxia Wang received her M.Eng. degree in circuit and system from the East China University of Technology at Fuzhou in 2009. She is currently a Ph.D. candidate at Shanghai University, Shanghai, China. Her current research interests include distributed machine learning, parallel and distributed computing.

Yongmei Lei received her Sc.D. degree in 1999 from Xi'an Jiaotong University at Xi'an, and her M.Eng. degree in 1989 from Xidian University at Xi'an. She is Professor at School of Computer Engineering and Science of Shanghai University. She has published over 30 technical papers. Her main research interests include parallel and distributed computing, big data analysis, etc.



Shenghong Jiang received her B.Sc. degree in electronics and information engineering from Anhui Science and Technology University, Anhui in 2016. She is currently a postgraduate student at Shanghai University, Shanghai, China. Her current research interests include parallel computing, distributed machine learning and high performance computing.