Robust Searching-based Gradient Collaborative Management in Intelligent Transportation System

HONGJIAN SHI, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China
HAO WANG, Division of Computer Science and Engineering, Louisiana State University, USA
RUHUI MA*, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China
YANG HUA, School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfas, UK
TAO SONG, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China
HONGHAO GAO, School of Computer Engineering and Science, Shanghai University, China
HAIBING GUAN, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China

With the rapid development of big data and the Internet of Things(IoT), traffic data from an Intelligent Transportation System(ITS) is becoming more and more accessible. To understand and simulate the traffic patterns from the traffic data, Multimedia Cognitive Computing(MCC) is an efficient and practical approach. Distributed Machine Learning(DML) has been the trend to provide sufficient computing resources and efficiency for MCC tasks to handle massive data and complex models. DML can speed up computation with those computing resources but introduces communication overhead. Gradient collaborative management or gradient aggregation in DML for MCC tasks is a critical task. An efficient managing algorithm of the communication schedules for gradient aggregation in ITS can improve the performance of MCC tasks. However, existing communication schedules typically rely on specific physical connection matrices, which have low robustness when a malfunction occurs. In this paper, we propose Robust Searching-based Gradient Collaborative Management(RSGCM) in Intelligent Transportation System, a practical ring-based gradient managing algorithm for communication schedules across devices to deal with ITS malfunction. RSGCM provides solutions of communication schedules to various kinds of connection matrices with an acceptable amount of training time. Our experimental results have shown that RSGCM can deal with more

*Ruhui Ma is the corresponding author.

Authors' addresses: Hongjian Shi, shhjwu5@sjtu.edu.cn, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800, Dongchuan Rd., Shanghai, China, 200240; Hao Wang, haowang@lsu.edu, Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, Louisiana, USA; Ruhui Ma, ruhuima@sjtu.edu.cn, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800, Dongchuan Rd., Shanghai, China, 200240; Yang Hua, Y.Hua@qub.ac.uk, School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfas, Belfast, UK; Tao Song, songt333@sjtu.edu.cn, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800, Dongchuan Rd., Shanghai, China, 200240; Honghao Gao, gaohonghao@shu.edu.cn, School of Computer Engineering and Science, Shanghai University, Shanghai, China; Haibing Guan, hbguan@sjtu.edu.cn, School or Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800, Dongchuan Rd., Shanghai, China, 200240.

1551-6857/2022/7-ART $15.00
https://doi.org/10.1145/3549939

varieties of connection matrices than existing state-of-the-art communication schedules. RSGCM also increases the robustness of ITS since it can restore the system's functionality in an acceptable time when device or connection breakdown happens.

CCS Concepts: • **Computer systems organization → Fault-tolerant network topologies**; • **Computing methodologies** → *Multi-agent planning*; • **Networks** → Cloud computing.

Additional Key Words and Phrases: all-reduce, communication scheduling, gradient aggregation, robustness, collaborative management

1 INTRODUCTION

The rapidly increasing intelligence of Intelligent Transportation System(ITS)[37, 46] and the increasing volumes of traffic data have turned traffic data processing and analyzing into Multimedia Cognitive Computing(MCC)[1, 21, 60, 69, 71, 73, 75]. To derive the traffic patterns and human behaviors from the massive data collected from ITS, the demand for more efficient and effective computing ability has been the main problem in ITS. Similar to traditional AI tasks, MCC for ITS can also adopt Distributed Machine Learning(DML) techniques to provide sufficient computing resources to handle the collected traffic data.

A critical issue for ITS to fully utilize the advantages and the capacity of available computing resources is the collaborative management of data. Although using more devices provides more computation power and reduces the computation time, it also introduces additional communication costs, which slows down the tasks. Carefully designing data managing strategies for different computing resources conditions and applications is critical for MCC tasks in ITS. The MCC tasks in ITS are closely related to DML for AI tasks, so MCC in ITS can borrow the techniques from traditional DML, with some proper modifications. One way for DML to better utilize the computational resources is through parallelism. Parallelism allows the devices to perform computation simultaneously, but with little communication between devices to ensure the correctness of the computation. As part of the collaborative management of devices, parallel communication is essential to increase efficiency and robustness.

DML exploits parallelism across multiple computing devices to accelerate the training process. There are two major parallelism schemes, model parallelism, and data parallelism. Model parallelism divides a model into sub-models and assigns them to different devices, which is helpful to handle huge deep learning models with millions of parameters like VGG[56], DenseNet[27], and BERT[15]. Instead, data parallelism splits a dataset into sub-datasets to process with different devices to tackle the tasks with large datasets, such as ImageNet[48]. In model parallelism, a sub-model usually takes the output of its precedent sub-models as input, which brings dependency between the two sub-models. Such dependency makes it hard to fully parallelize the computation of all sub-models. However, data parallelism does not enforce such dependencies so that each device can execute forward propagation and backpropagation simultaneously. Since the dataset can be shared among different deep learning models, data parallelism is relatively easy and can be used universally on different computing devices. In addition, due to the heterogeneity of devices and connections in ITS, data parallelism is a better implementation than model parallelism, so it is more suitable for MCC tasks in ITS.

Data parallelism extensively reduces the computation time, but the communication cost rises as the device boosts. With more devices participating in training, the collaborative management of data has been a critical issue in ITS. One important scenario is the collaborative management of gradients in MCC tasks in ITS. The cost from gradient aggregation[65] across devices dominates the whole training process during the training process. [10] proves that the performance of Synchronous Stochastic Gradient Descent (SSGD) is better than the performance of Asynchronous Stochastic Gradient Descent (ASGD), we mainly focuses on the optimization of the SSGD in this paper. SSGD first generates a global gradient using the local gradients on all devices, and then returns the global gradient back to all devices. All-reduce[58] is a widely applied collective for gradient aggregation in DML. Gradient aggregation with all-reduce collective has been developed in the past decades, and many communication

schedules have been introduced. In the beginning, originate from the idea of parameter server[34], the approaches are centralized designs, also called the tree-based communication schedules, like [58, 74]. In tree structures, the root device is responsible for aggregating all the local gradients, computing the global gradient, and sending the global gradient back to the other devices. However, the heterogeneous resource demands of tree structures between root device and other devices lead to an uneven workload distribution on different devices, which results in limited system scalability. As the number of devices increases, the urge to develop a high scalability structure drives the emergence of decentralized designs, also called the ring-based communication schedules. Such communication schedules have an even workload on each device, so the scalability improves[3].In ITS, communication is usually limited, where only a small portion of data can be transferred simultaneously to a device. In such cases, ring-based communication schedules, which require less data transition, are better than tree-based communication schedules.

Most research focuses on the efficiency of gradient aggregation by modifying the communication schedule and the physical connection matrix. However, another critical issue is the robustness of ITS. As a practical and real-time scenario, device malfunction or connection malfunction is a possible event in ITS. Maintaining the functionality of the whole system when facing those conditions needs to be carefully addressed. Currently, for gradient aggregation in MCC, most of the implementations have a specific physical connection matrix that supports the communication schedule, which means that if one of the devices or connections breakdown, the communication schedule collapses. In addition, none of the implementations includes a backup communication schedule, which largely reduces the robustness of gradient aggregation. There were researches on network collaboration like [8, 39, 40, 64, 70, 72], on network robustness like [2, 12, 23, 24, 47, 57, 67, 76], and on DML like [4–6, 11, 25, 32, 36, 61–63]. Among all those researches, Blink[61] is the only recent algorithm that increases the robustness of gradient aggregation in DML. Although Blink[61] was proposed as a gradient managing algorithm, it mainly focuses on finding a more efficient communication schedule with spanning trees. As tree-based communication schedules fail to handle communication-limited scenarios, such tree-based communication scheduling algorithms like Blink can hardly be used in our scenarios. Still, current applications like MPI[20], NCCL[44], Horovod[52], TensorFlow[22], and PyTorch[19] adopt the ring-based communication schedules, so a ring-based communication scheduling algorithm is needed.

In this paper, we propose Robust Searching-based Gradient Collaborative Management(RSGCM) in Intelligent Transportation System, a ring-based gradient managing algorithm that can provide communication schedules for gradient aggregation for the collaborative management in ITS if device or connection malfunction happens. RSGCM requires zero modification on the physical level, which decreases the reliability of the gradient aggregation on the physical condition, thus increasing the robustness of gradient aggregation. With RSGCM, even if the MCC tasks in ITS suffer from a device or connection breakdown, the training can continue. Our contributions include:

- We propose RSGCM, a communication scheduling algorithm for ring-based communication schedules that can increase the robustness of gradient aggregation. RSGCM uses Monte Carlo Tree Search as the optimization method, which proves to be the most stable optimization method.
- We have formulated gradient aggregation scheduling problem with mathematical representation that can be solved or optimized and presented most SOTA communication schedules with the formulation.
- We have evaluated RSGCM on its robustness and parameter representative ability. The experimental results show that RSGCM can adapt to different connection matrices or breakdown conditions that other communication schedules fail to handle and also show the effectiveness of Monte Carlo Tree Search.

2 BACKGROUND AND MOTIVATION

Data parallelism splits the global dataset into local datasets and assigns them to multiple computing devices. During each iteration, data parallelism uses a three-step training procedure. First, each device performs the forward

propagation on its local dataset to obtain the local gradient. Second, gradient aggregation is conducted among all devices to compute the global gradient using the local gradients. Third, each device performs backpropagation using the global gradient and updates the original model. Since each device requires the global gradient computed by the local gradients, the collective to use is all-reduce.

The all-reduce collective has been developed in High Performance Computing for a long time but was introduced to DML in the last decade. The development of gradient aggregation with the all-reduce collective, or communication schedules, is shown in Fig. 1.

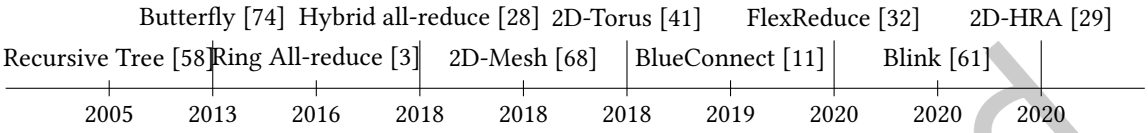


Fig. 1. The timeline of the development of gradient aggregation algorithms with the all-reduce collective.

Communication schedules mainly contain two kinds, tree-based and ring-based. Tree-based communication schedules are centralized methods. There are two major tree structures, Recursive Tree[58] and Butterfly[74]. Ring-based communication schedules are decentralized methods. It starts from Baidu's Ring all-reduce[3], widely used in MPI[20], NCCL[44], and Horovod[52]. NCCL is used both in TensorFlow[22] and PyTorch[19]. The workload of Baidu Ring all-reduce is even among all devices, and those gradient portions, or data blocks, can be transferred independently to increase the efficiency. But the scalability decreases as more devices are included. More devices bring more communication steps, which increases the latency of the communication schedule. Other ring-based communication schedules are proposed to tackle this problem, and most of them use hierarchical approaches, like Hybrid all-reduce[28], 2D-Mesh[68], 2D-Torus[41], and 2D-HRA[29]. A typical communication schedule is 2D-Mesh[68] shown in Figure 2. The first step is a horizontal ring all-reduce on the first half of each data block and a vertical ring all-reduce on the second half of each data block. The second step is the reverse of the first step: a horizontal ring all-reduce on the second half of each data block and a vertical ring all-reduce on the first half of each data block. This completes the communication schedule. This indicates that most communication schedules can be divided into multiple ring all-reduces.

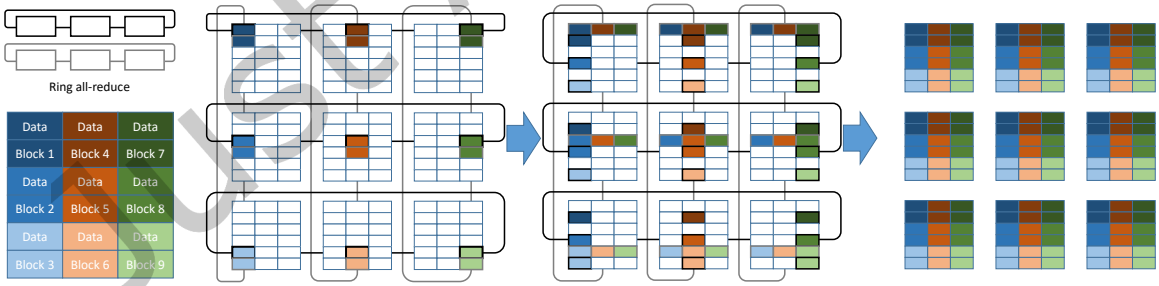


Fig. 2. Communication schedule of 2D-Mesh[68]

However, those communication schedules need specific physical connection matrices to support the transition of data, which means that they have low robustness. That is, a specific physical connection or a specific device breakdown leads the whole schedule to malfunction. Blink[61] provides a gradient managing algorithm for different connection matrices based on spanning trees for tree-based communication schedules. No gradient managing algorithm can handle different connection matrices for ring-based communication schedules.

3 ROBUST SEARCHING-BASED GRADIENT AGGREGATION(RSGCM)

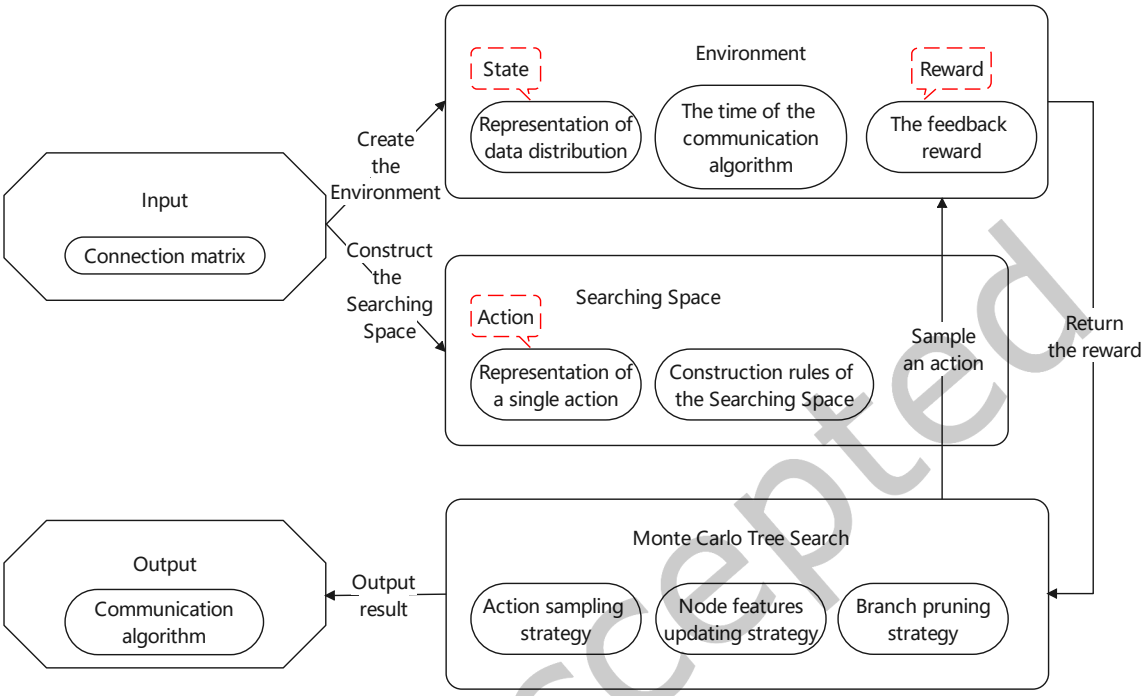


Fig. 3. Overview structure of our system.

We propose Robust Searching-based Gradient Collaborative Management(RSGCM) in Intelligent Transportation System, a gradient managing algorithm for ring-based communication schedules that increases the robustness of the gradient aggregation process. Our system takes the physical connection matrix as the input and outputs the communication schedule. The physical connection matrix describes whether there is a connection between two devices. Two values are transferred between the three sub-algorithms, the action and the reward. An action includes a set of ring all-reduces that can be performed simultaneously. A reward represents how well is the chosen action.

The main structure of RSGCM includes three sub-algorithms shown in Fig. 3. The first is the Searching Space, a set of actions to be sampled. The major designs in this sub-algorithm are the representation of a single action and the construction rules of the Searching Space. The second is the Monte Carlo Tree Search (MCTS), the decision-making agent that chooses an action from the Searching Space with a specific strategy to perform in the environment. The major designs in this sub-algorithm are the strategy to choose action, the updating strategy of tree nodes, and the pruning strategy of the tree. The third is the Environment, a sub-algorithm that takes the action chosen from the MCTS and evaluates its performance. The feedback from Environment is used to update the MCTS for better choice. The major designs in this sub-algorithm are the representation of the data distribution, the simulation model, the actual implementation, and the design of the reward.

In conclusion, RSGCM takes a physical connection matrix and constructs the Searching Space. Then it repeatedly samples an action from the Searching Space, evaluates the received action in the Environment to get the reward, and updates the Monte Carlo Tree. Finally, the Environment provides the final communication schedule as the

result. The following sections are arranged as follows. Section 3.1 introduces the Searching Space; section 3.2 introduces the MCTS; section 3.3 introduces the Environment.

3.1 Searching Space

The first sub-algorithm is the Searching Space. The actions are sampled from Searching Space. It mainly has two critical designs, the representation of a single action and the construction rules of the Searching Space. The notations used in this section are shown in Table 1.

Table 1. The notations for Searching Space

Notations	Meanings	Notations	Meanings	Notations	Meanings
r	list of rings-set	G	connection matrix	f	number of devices in a ring
r_i	a set of rings	m	maximum connection	I	initial device for generating
r_{ij}	a single ring	S	system parameter	N	total number of devices
b	a list of blocks	P	connection priority	DNL	device number list
b_i	a single block	E	extension parameter	DAM	device available matrix
FL	factor list	U	searching upper limit	CAM	connection available matrix

For the representation of a single action, we propose the following format. Each action contains two parts, the list of ring-sets r and the corresponding list of blocks b. In r, there are several ring-sets $r_1, r_2, ..., r_s, ...$, where each ring-set contains several rings $r_{s1}, r_{s2}, ..., r_{si}, ...$ All the rings in r can be performed simultaneously in the Environment because they don't share the connections. All the rings in r_s can be performed on the same data block since they don't share the devices. Each block b_s in b is corresponding to r_s in r. As a result, the representation of a single action is shown in (1).

$$r = [r_1, r_2, ..., r_s, ...] = [[r_{11}, r_{12}, ..., r_{1i}, ...], [r_{21}, r_{22}, ...], ...] \tag{1}$$

$$b = [b_1, b_2, ..., b_s, ...] \tag{2}$$

We can describe this action as: all rings in r_s perform ring all-reduce on data block b_s.

The construction of Searching Space includes several episodes, and each episode results in an action in the Searching Space. Several steps are needed in an episode. To construct the Searching Space, we here introduce our construction rules. The idea is taking the connection matrix G as the input, together with the system parameter S, iteratively finding different actions according to different iteration-related parameters, including the number of devices in a ring f, the initial device I, and connection priority P. The system parameter S represents the hardware capability. $S = 1$ means that a single device can support one sending process and one receiving process simultaneously, same as the device capability used in 2D-Torus[41]. $S = 2$ implies that a single device can simultaneously support two sending processes and two receiving processes, which is the TPU capability[31] used in 2D-Mesh[68]. We only consider those two S since other values don't reflect actual device capability.

Here we further explain the iteration-related parameters. f is the number of devices in a single ring that we will search for in one construction episode. We first get the total number of devices N. Then we set a extension parameter E to obtain a device number list $DNL = [N, N + 1, ..., N + E]$. Next with a upper limit U, calculate the factor of all elements in DNL that is less than U, we can construct a factor list $FL = [f_1, f_2, ...]$ satisfies that $3 \leqslant f_i < U$. Finally, the value of f is chosen from FL. I is the initial device with which our construction episode starts. The value of I is the device's index, which is chosen from $[0, 1, ..., N - 1]$. The purpose of I is to ensure that all devices can appear in an action so that the communication schedule can complete the all-reduce collective. P

Algorithm 1 Action Construction Algorithm

Input: Connection matrix G, system parameter S, number of devices in a ring f, initial device I, and connection priority P

Output: A single action A

Initialize $A = [r, b]$ where $r = [], b = []$, and $DAM = CAM = G$.
for Ring-set $s = [0, 1, ..., S-1]$ **do**
  Initialize $r_s = []$.
  while there are more than f devices in DAM and there are possible rings that haven't been constructed **do**
    Initialize $r_{si} = [I]$ and the current processing device $K = I$.
    while The devices in r_{si} don't form a complete ring **do**
      Choose a device $K_{new} = Adj[K][P] = K_P$ from the adjacency list $Adj[K] = [K_1, K_2, ...]$of device K
      Add K_{new} to r_{si}.
      if r_{si} forms a complete ring of f device **then**
        $r_s.append(r_{si})$.
        From DAM, delete all devices exists in r_{si}.
        From CAM, delete all connections exists in r_{si}.
      else if r_{si} has f devices but doesn't form a complete ring **then**
        Remove K_{new} from r_{si}.
        Set $P = (P+1)\%m$.
      else if r_{si} doesn't have f devices **then**
        Set $K = K_{new}$.
      end if
    end while
  end while
  Choose one of the block operations b_s.
  $r.append(r_s), b.append(b_s)$.
end for
The final action is $A = [r, b]$.

is the connection priority we used to choose a connection from the adjacency list of a single device. If a single device can connect to up to m devices in the connection matrix, then the value of P is $[0, 1, ..., m-1]$ indicates the index of the next device in the current device adjacency list.

We can perform the action construction algorithm with those iteration-related parameters as in Algorithm 1. In the algorithm, we maintain the device availability matrix DAM and the connection availability matrix CAM, indicating the device and connection available later.

Using Algorithm 1, we can construct a single action, and through multiple episodes, we can construct the whole Searching Space. As a result, a 2-dimension mesh grid connection matrix in 2D-Mesh constructs a Searching Space of 445 actions, an acceptable sample size for the MCTS.

To better get through the procedure of a single action's construction, we here provides an example. Suppose we are construct with G in Fig. 4, $m = 3$, $S = 2$, $f = 4$, $I = 1$, and $P = 2$. We are to obtain $A = [r, b] = [[r_0, r_1], [b_0, b_1]]$. The process is shown in Figure 4.

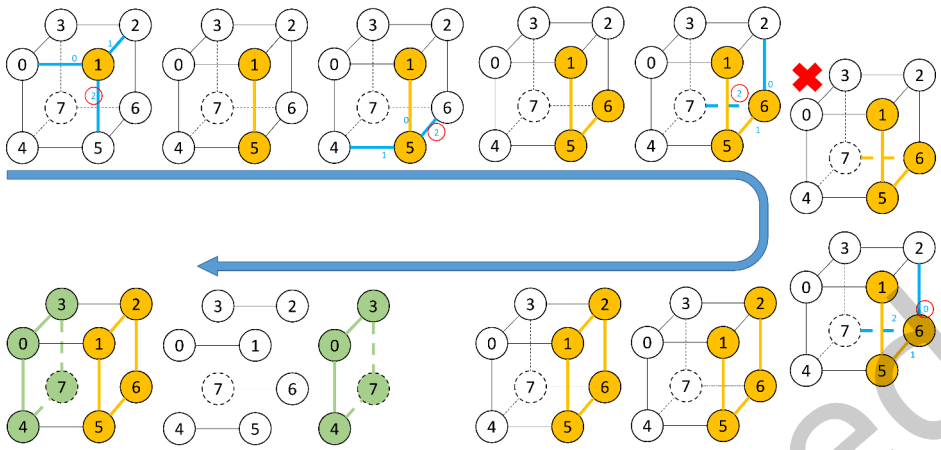


Fig. 4. The process of creating an action. First, starting from node $I = 1$, the agent finds node 5 of index $P = 2$ from node 1's adjacency list. Second, the agent finds node 6 of index $P = 2$ from node 5's adjacency list. Third, the agent finds node 7 of index $P = 2$ from node 6's adjacency list, but the $f = 4$ nodes do not form a ring. So in the fourth step, the agent finds node 2 of index $P' = 0$ from node 6's adjacency list. Thus $[1, 5, 6, 2]$ forms a ring. Remove them from DAM and CAM, and start looking for another ring. $[3, 7, 4, 0]$ forms another ring. After that, all nodes are removed, so r_0 has been constructed. Next, find another set of ring r_1 from CAM. Finally we can obtain this action of $r = [[[1, 5, 6, 2], [3, 7, 4, 0]], []]$.

We can find a solution to the all-reduce collective through the construction procedure, but it can be a sub-optimal solution instead of an optimal solution. Our Searching Space construction procedure decreases the searching time for the solution at the cost of reducing the probability of finding the optimal solution. In addition, although the Searching Space only contains ring structures after the construction procedure, we can also expand the Searching Space by adding tree structures to it. In this way, the gradient managing algorithm can find a more efficient communication schedule but can take longer to find the solution.

3.2 Monte Carlo Tree Search (MCTS)

Our second sub-algorithm is the Monte Carlo Tree Search (MCTS) for sampling actions. From Section 3.1, we have already reduced the size M of the Searching Space, but we are still dealing with a huge searching space, so we need an efficient agent to perform the sampling and updating processes. For a Searching Space constructed from a connection matrix of 9 devices, we are sampling from $M = 445$ actions.

Several algorithms have been used for different optimization problems, but such algorithms are hard to implement on such searching space. For traditional algorithms, we've considered the enumeration algorithm. Enumeration algorithm[14] takes unacceptable time to search for the solution since there are too many possibilities. If computing the optimal solution directly, solvers[7] usually obtain the best performance. Solvers find the exact solution to an optimization problem, but the computation complexity is relatively large. Although solvers can be used on such problems, it is not suitable in the case of DML robustness instead of efficiency. For heuristic algorithms, we've considered greedy algorithm, Hill Climbing(HC), Simulated Annealing(SA), Particle Swarm Optimization(PSO), and Ant Colony Optimization(ACO). The greedy algorithm[14] fails because the feedback only exists when the all-reduce collective is finished, but not from each communication step. HC[51] is an upgrade of greedy algorithm that chooses the optimal solution from the neighbors of the current solution. It is likely to converge to local optimal, so it is not suitable for this problem. SA[18] is similar to MCTS, which is also based

on Monte Carlo sampling and exploration. The disadvantage of SA is that it does not fully utilize the history trajectories and does not maintain the exploration aspect after it converges to a sub-optimal. PSO[13] is originated from bird foraging. It is not applicable in this scenario because the optimization of gradient aggregation is not in continuous space with precise action dimensions, so it isn't easy to define the particle's position, and velocity as PSO requires. ACO[17] uses the trajectories of ants as the solutions, and the best solution is the shortest trajectory. It is similar to MCTS since it evaluates trajectories according to previous visits and finally converges to the best trajectory. MCTS is more likely an upgrade of ACO by its exploration aspects. A new approach for this kind of problem is Reinforcement Learning (RL), like REINFORCE[66], DDPG[35], A3C[42], and PPO[50]. All those algorithms failed to process searching space at this size, and feedback is sparse. Due to the lack of convergence and lengthy training time.

We then focus on MCTS, a heuristic algorithm that maintains the exploration aspect and has high robustness in our scenario. MCTS introduces a trade-off between training time and results to deal with a huge searching space while remaining time-efficient. It is originated from Monte Carlo sampling. The simple Monte Carlo sampling actions from the Searching Space with complete randomness and conclude the performance of each action according to the sampling history. The problem is that Monte Carlo sampling does not have sampling priority over better actions, which can mitigate the impact of good actions over the whole algorithm. MCTS maintains a tree structure to store the useful information from the previous sampling trajectories and has a sampling strategy that prefers to sample the trajectories that previously obtained better performance. Meantime, the strategy also keeps an exploration aspect that can be flexibly adjusted. With the exploration aspect, during sampling, better trajectories can still be found even though MCTS has converged. As a result, we choose MCTS as our optimization algorithm.

For MCTS, tree nodes and tree branches are two aspects. A single tree node in the MCTS represents a state or data distribution in the Environment. The decision of MCTS is based on this aspect. Several features are stored in a tree node, including the following, which are also shown in Table 2.

Table 2. The notations for MCTS

Notations	Meanings	Notations	Meanings	Notations	Meanings
M	searching space size	λ	exploration ratio	n	visiting counter
W	total reward	v	recent reward	R	average reward

- The exploration ratio λ indicates the probability of randomly choosing a specific child node from the current tree node. The initial value is $\lambda = \frac{1}{M}$.
- The visiting counter n indicates the total count that Environment went through certain branches and arrived at the current tree node. The initial value is $n = 0$.
- The total reward W indicates the total reward gotten in the previous visits. The initial value is $W = 0$.
- The recent reward v indicates the reward gotten from the last training episode.
- The average reward R indicates the average reward gotten in the previous visits. Its value is $R = \frac{W}{n}$.

A single branch in the MCTS represents an action or a communication step from the Searching Space. The format of an action is described in Section 3.1. There are three major designs, the strategy for sampling actions from the Searching Space, the strategy to update the features in the tree node, and the strategy to prune the branches that is no longer efficient.

First, for the strategy for sampling action, we adopt the formula used in AlphaGo[49, 53–55]. The formula to sample action is shown in (3). In the equation, $children[i]$ refers to the i^{th} child node of the current node and $probability[i]$ refers to the probability to choose the action to the i^{th} child node. The exploration ratio decides

the ratio of experience probability and exploration probability. The experience probability is the first term in (3) whose value is based on previous training episodes. The exploration probability is the second term in (3) whose value depends on the number of children nodes. If λ increases, it means that the agent is more likely to make decisions based on previous experience, where a larger average reward results in a larger sampling probability. If λ decreases, it means that the agent is more likely to choose actions that haven't been visited.

$$probability[i] = children[i].R + \lambda * children[i].p * \frac{\sqrt{n}}{1 + children[i].n} \tag{3}$$

Second, to update the node features, we also adopt the strategy used in AlphaGo[49, 53–55]. Equation (4) shows the updating strategy for each value. The idea of updating is to improve the MCTS to sample more efficient actions.

$$\begin{cases} n = n + 1 \\ W = W + v \\ R = \frac{W}{n} \end{cases} \tag{4}$$

The third is the pruning strategy. To reduce the redundant training steps, we need to reduce the size of the MCTS to obtain more efficient sampling. Also, the large size of MCTS introduces enormous memory stress to the devices, so reducing the number of branches is essential. There are three pruning strategies.

- If the current communication schedule in the episode already has worse performance than the best communication schedule at the current training step, we prune all the tree nodes under the current node.
- If the data transferred with the chosen action is redundant at the current training step, we prune all the tree nodes under the current node.
- If the agent already met the maximum training step in a single training episode at the current training step, we prune the current node.

3.3 Environment

The third sub-algorithm is the Environment for evaluating the actions chosen by the MCTS in Section 3.2. Environment can store the best communication schedule over the previous training episodes and give feedback to the MCTS. The Environment has two kinds, the Virtual Environment (VE) and the Realistic Environment (RE).

VE performs the training on a given model, which evaluates the action with a performance formula. VE simulates the data transition through a 3-dimension matrix called the data distribution or state. In that matrix, $data[i][j][k]$ indicates whether device i has received the data block k from device j. Under the above representation, the data distribution before and after the all-reduce collective are shown in (5) and (6)

$$i = j \Longleftrightarrow data[i][j][k] = 1 \tag{5}$$

$$\forall i, j, k \Longrightarrow data[i][j][k] = 1 \tag{6}$$

We here introduce the performance model to compute the reward of an action. Since our top concern is to speed up the communication schedule, our performance model derives the reward from the execution time of the communication schedule. During the all-reduce collective, the computation time to reduce the data only depends on the computational power of a single device, while the communication time is related to the communication schedule, we only evaluate the communication time using (α, β)[26] model where α is the latency τ_L and β is the bandwidth τ_B. Using the notations in Table 3, we can represent the model with (7).

Table 3. The notations for Environment

Notations	Meanings	Notations	Meanings	Notations	Meanings
α,τ_L	connection latency	β,τ_B	connection bandwidth	D	transferred data

$$\tau = \tau_L + \tau_B * D \tag{7}$$

Using (7), we are able to evaluate the performance of State-of-the-art (SOTA) communication schedules shown in Table 4. We use $T_{t-a}(D, N)$ to denote the cost ($t = L$ means latency cost, $t = B$ means bandwidth cost) of certain unit communication schedule ($a = b$ means Butterfly[74], $a = r$ means Baidu Ring all-reduce[3]) on a cluster of N devices and total amount of data with size D.

In Table 4, i represents the number of devices in a row, and j represents the number of devices in a column in 2D-Mesh[68] and 2D-Torus[41]. u represents the device number in each small group of Hybrid all-reduce[28]. Also, to simplify the total cost, we adopt several simplification functions in (8).

$$F_1(D, N) = 2\tau_L N - \frac{2\tau_B D}{N} \tag{8}$$

$$F_2(D) = \tau_L + \tau_B D \tag{9}$$

$$F_3(D) = \tau_L - \tau_B D \tag{10}$$

Table 4. The cost of SOTA communication schedules under our performance model

Schedules	Latency Cost	Bandwidth Cost	Total Cost
Butterfly[74]	$T_{L-b}(D, N) = \tau_s \lceil \log_2 N \rceil$	$T_{B-b}(D, N) = \tau_c D \lceil \log_2 N \rceil$	$F_2(D) \lceil \log_2 N \rceil$
Ring all-reduce[3]	$T_{L-r}(D, N) = 2\tau_s(N-1)$	$T_{B-r}(D, N) = 2\tau_c(N-1)\frac{D}{N}$	$F_1(D, N) - 2F_3(D)$
Recursive tree[58]	$2T_{L-b}(D, N)$	$2T_{B-b}(D, N)$	$2F_2(D) \lceil \log_2 N \rceil$
Hybrid all-reduce[28]	$\sum_{k=u,N/u} T_{L-r}(D, k) + \tau_s$	$\sum_{k=u,N/u} T_{B-r}(D, k) + \tau_c D$	$\sum_{k=u,N/u} F_1(D, k) - 4F_3(D) + F_2(D)$
2D-Torus[41]	$T_{L-r}(D, i) + T_{L-r}(D, j)$	$T_{B-r}(D, i) + T_{B-r}(D, j)$	$F_1(D, i) + F_1(D, j) - 4F_3(D)$
2D-Mesh[68]	$2 \max_{k=i,j}(T_{L-r}(\frac{D}{2}, k))$	$2 \max_{k=i,j}(T_{B-r}(\frac{D}{2}, k))$	$2F_1(\frac{D}{2}, \max(i, j)) - 4F_3(\frac{D}{2})$
2D-HRA[29]	$\sum_{k=i,j,u} (T_{L-r}(D, k)) + \tau_s$	$\sum_{k=i,j,u} (T_{B-r}(D, k)) + \tau_c * D$	$\sum_{k=i,j,u} (F_1(D, k)) - 6F_3(D) + F_2$

Our performance model is capable of representing all SOTA communication schedules mentioned. Based on that performance model, we can get the simulation time of the communication schedule in each training episode and save the best communication schedule with the least simulation time in the Environment. According to that simulation time, we can derive the feedback from the VE to the MCTS. The feedback consists of three parts.

- The done condition is whether the Environment has completed the all-reduce collective. According to the data distribution after all-reduce collective in (6), when that distribution is met, we determine that the all-reduce collective is done and is ready for feedback and update.
- The valid data transferred indicate whether the current action successfully transferred data. If there is no data transmitted, the VE will provide that information to the MCTS for pruning.
- The reward is the value used to update the MCTS nodes. We use the following steps to obtain the corresponding reward from the simulation time. We maintain a reward list. There is an initial 0 in the list, which means that the communication schedules that don't finish the all-reduce collective get a reward of 0. If an episode finishes the all-reduce collective during the training, it inserts the communication schedule's simulation time to the reward list in descending order. The index is the reward that will provide to the MCTS. For example, a previous reward list is $[0, 8.3, 6.5, 4.7]$. If the current episode results in a communication schedule with a simulation time of 6.5, then it will receive a reward of 2. Suppose the current episode results in a communication schedule with a simulation time of 5.4. In that case, the new time is inserted into the reward list so that the reward list is $[0, 8.3, 6.5, 5.4, 4.7]$, and the current episode gets a reward of 3.

RE performs the training on an existing system, which evaluates the action with the system's feedback. The system performs communication schedules with NCCL[44] and CUDA[45] on a cluster of devices. We derive a list of sets of ring all-reduce from the Environment as the representation of the communication schedules. Each ring corresponds to a ncclAllReduce function call to perform the data transition in the actual system.

The data distribution is represented by the data buffer in the actual system. All communication is inplace communication, which means the sending buffer and the receiving buffer have the same address. The data distribution before and after the all-reduce collective are represented by (11) and (12) where $buffer[i]$ indicates the data on device i.

$$buffer[i] = [DATA_{i1}, DATA_{i2}, ...] \tag{11}$$

$$buffer[i] = [\sum_{j=0}^{N-1} DATA_{j1}, \sum_{j=0}^{N-1} DATA_{j2}, ...] \tag{12}$$

The time of the communication schedule T is measured using cudaEvent, whose unit is millisecond and the precision is a microsecond. Since the data size is D and the device number is N, the valid data transferred during the all-reduce collective is $D * (N-1) * N$, because the initial data on each device have been sent to all other $N-1$ devices. Thus, our reward for the existing system is defined with (13).

$$reward = \frac{D * (N-1) * N}{T} \tag{13}$$

In addition, our Environment aims at completing the all-reduce collective, but it is also capable of dealing with other collectives like reduce-scatter, all-gather, broadcast, etc. Using all-reduce collective in our problem is that the data distribution pattern of gradient aggregation is the same as all-reduce.

4 EVALUATION

RSGCM aims at increasing the robustness of the gradient aggregation process through a gradient managing algorithm for the communication schedules that can handle malfunction in DML system. In this section, we provide the settings of hyperparameters in Section 4.1, the major result that proves the robustness of the algorithm in Section 4.2, and the influence of the modification on different parameters in Section 4.3.

In addition, to better describe the resulting communication schedules, we here introduce the term $ring_{f,z}^{S,w}$. In that expression, f is the number of devices in a single ring, S is the system parameter, z is the total number of rings in that action, and w is the number of data blocks that those rings are operating on whose maximum value is S. Then those SOTA ring-based communication schedules can be expressed in Table 5 where the notations are the same as in Table 4.

Table 5. The expression of SOTA ring-structured GPU communication schedules

SOTA schedules	Expression
Ring all-reduce[3]	$ring_{N*1}^{1,1}$
Hybrid all-reduce[28]	$ring_{u,N/u}^{1,1} + ring_{N/u,1}^{1,1} + ring_{u,N/u}^{1,1}$
2D-Torus[41]	$ring_{i,j}^{1,1} + ring_{j,i}^{1,1}$
2D-Mesh[68]	$ring_{\sqrt{N},\sqrt{N}}^{2,1} + ring_{\sqrt{N},\sqrt{N}}^{2,1}$
2D-HRA[29]	$ring_{u,N/u}^{1,1} + ring_{i,j}^{1,1} + ring_{j,i}^{1,1} + ring_{u,N/u}^{1,1}$

4.1 Settings of hyperparameters

We first identify the meaning and initial values of the hyperparameters used during the experiments.

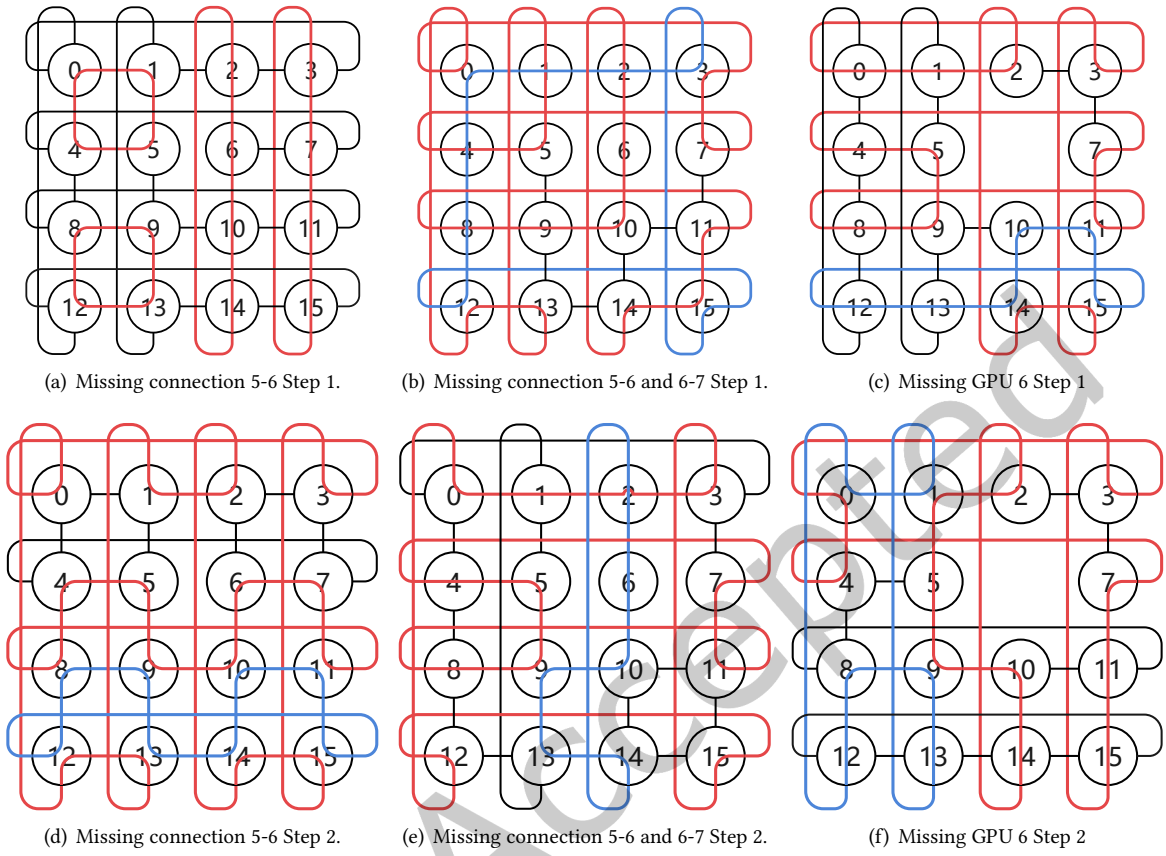
- The input connection matrix G is initially set to the 2-dimension mesh grid matrix used in 2D-Torus[41] and 2D-Mesh[68]. The number of devices on the 2 dimensions is equal, which means that the connection matrix for 16 devices is a $4 * 4$ matrix and the connection matrix for 25 devices is a $5 * 5$ matrix.
- The simulation latency is $\tau_L = 9\mu s$, and the bandwidth is $\tau_B = 39\mu s/MB$. Those parameters are derived from the feature of NVLink[33].
- The total number of episodes Ep indicates the number of communication schedules that we will sample during the training process. The larger the value, the higher the probability of finding the optimal solution; the lower the value, the quicker the training process. Initially, we set $Ep = 8 * M$.
- The total number of sampling steps St in an episode indicates the maximum actions in a communication schedule. Initially, we set $St = N$, which ensures the existence of a communication schedule from the Searching Space that can complete the all-reduce collective.
- The exploration ratio λ indicates how much the agent makes decisions based on experience or exploration. The value of λ follows a descending strategy in (14). In the equation, $\lambda_1 = 10$ indicates the initial value of λ, $\lambda_2 = 0$ indicates the final value of λ, Ep_{ex} indicates the number of episodes used from λ_1 to λ_2, which also means the number of episodes that includes an exploration part, and Ep_c indicates the current episode number.

$$\lambda = \lambda_1 - \frac{\lambda_1 - \lambda_2}{Ep_{ex}} * Ep_c, \tag{14}$$

- The data size D indicates the total amount of data on each device that is going to perform the all-reduce collective. We set this value to 32MB for simulation.

4.2 Major experimental results

RSGCM can identify a communication schedule when the physical connection matrices of SOTA algorithms encounter malfunction of devices or connections. As shown in Fig. 5, with system parameter $S = 2$, latency



(a) Missing connection 5-6 Step 1. (b) Missing connection 5-6 and 6-7 Step 1. (c) Missing GPU 6 Step 1

(d) Missing connection 5-6 Step 2. (e) Missing connection 5-6 and 6-7 Step 2. (f) Missing GPU 6 Step 2

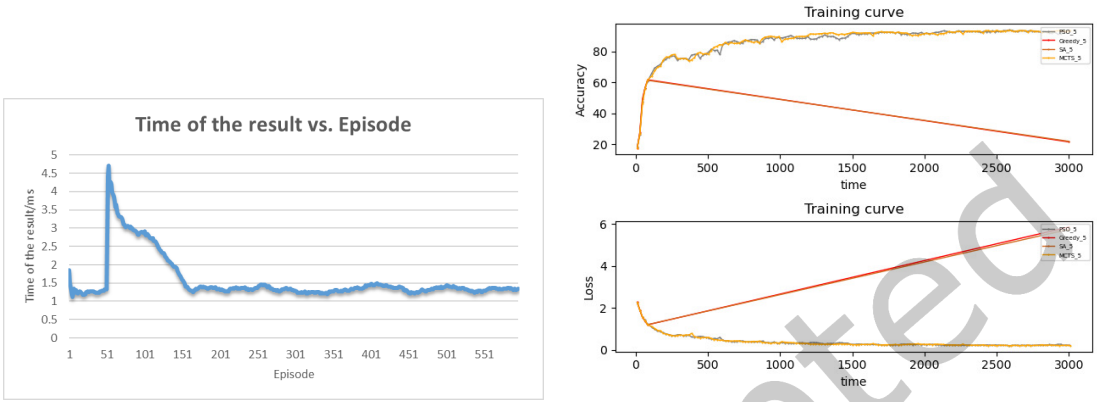
Fig. 5. The training results include exactly two steps for connection failure or device failure of the 2-dimension mesh grid connection matrix. Red lines indicate the rings in the first ring-set, and the blue lines indicate the rings in the second ring-set.

$\tau_L = 0.01ms$, and bandwidth $\tau_B = 0.1ms/MB$, it appears that with connection malfunction or device malfunction, RSGCM can find a communication schedule that can achieve the all-reduce collective.

We also test our gradient managing algorithm on an actual system of 8 devices with the connection matrix shown in Fig. 4. Our scenario is that at episode 50, if the connection between device 6 and 7 breakdowns, the agent will start functioning and try to identify the possible communication schedules to complete the all-reduce collective. The time to complete the all-reduce collective is shown in Fig. 6. We can observe that when the malfunction happens, the time used for the all-reduce collective suddenly boosts. Still, as the number of episodes increases, the solution found by the agent is gradually improving, which results in the figure that the time used is closer to the time used before the malfunction.

Also, we conducted an experiment on the image classification task of a 4-layer CNN over MNIST dataset, shown in Figure 6 with optimization methods including Greedy[14], Simulated Annealing(SA)[18], Particle Swarm Optimization(PSO)[13], and Monte Carlo Tree Search(MCTS)[49, 53–55]. The connection matrix follows the condition in Figure 5 where the connection 5-6 are missing. The data are unevenly distributed, where each device only contains 2 labels out of the 10 labels in MNIST. In that case, aggregation is necessary to complete

the training. The malfunction occurs in episode 5. Also, generating the Searching Space can start before the malfunction occurs. We can remove the unavailable actions from the Searching Space according to the condition and start optimizing. In this way, we can mitigate the overhead of generating the Searching Space.



(a) (b)

Fig. 6. (a) The time graph of the performance of the best communication schedule concerning the training episode. The failure happens in episode 50. (b) Training curve of MCC task with different optimization methods including Greedy, Simulated Annealing, Particle Swarm Optimization, and Monte Carlo Tree Search. "PSO_5" indicates the 5th client's training curve when using PSO.

The result shows that Greedy and SA failed to find a feasible solution for aggregation. Although PSO and MCTS can initialize a feasible solution and continue optimizing, MCTS is more stable than PSO. From the experiments shown in Figure 7, MCTS successfully initializes 5/5 feasible solutions while PSO fails to initialize 2/5 solutions.

Compared to SOTA communication schedules, our algorithm shows robustness over malfunction. No backup communication schedules or communication scheduling algorithms are provided for Baidu's Ring all-reduce[3], Hybrid all-reduce[28], 2D-Mesh[68], 2D-Torus[41], or 2D-HRA[29].

4.3 Influence of parameters

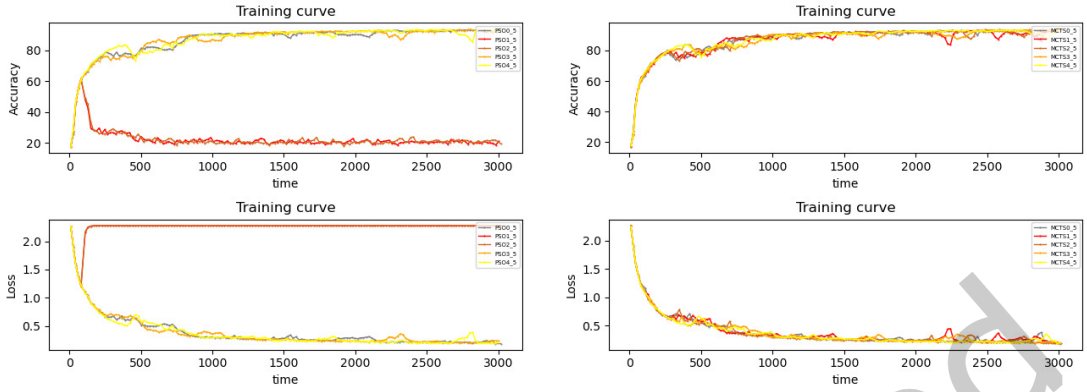
RSGCM can represent different connection matrices through modifying the parameters in RSGCM.

First is the system parameter S. The results with different S are shown in Fig. 8. We can find that when $S = 1$, each device is only included in one of the rings at a certain time, but when $S = 2$, each device can be included in at most two of the rings at a certain time. S represents the different hardware capabilities of devices used in 2D-Torus[41] and TPUs[31] in 2D-Mesh[68].

Second is the latency-bandwidth-ratio (LBR) defined in (15).

$$LBR = \frac{\tau_L}{\tau_B} \tag{15}$$

It represents the connection capability. Intuitively, when $S = 1$, 2D-Torus[41] should be better than Baidu Ring all-reduce[3] and when $S = 2$, 2D-Mesh[68] should be better than Double Ring all-reduce where there are two ring all-reduces, and each one of them operates on all devices and on one data blocks out of the two. However, from the result in Fig. 8, we find that all solutions are Baidu Ring all-reduce or Double Ring all-reduce. The reason is



(a) (b)

Fig. 7. (a) The training curve of the MCC task when using PSO. (b) The training curve of the MCC task when using MCTS. The malfunction occurs in episode 5. "PSO1_5" indicates the training curve of the 5th client in the 1st trial. The optimization runs for 5 separate trials to measure the stability. As a result, MCTS is more stable at finding a feasible solution than PSO.

that with different LBR, the latency or the bandwidth can either be the bottleneck of the communication schedule. In Table 6, we compared the solutions of the same connection matrices under $LBR = 10:1$ and $LBR = 1:10$. It appears that when $N = 9, LBR = 10:1$, the result is a 2D-Mesh, and when $N = 25$ or $LBR = 1:10$, the four results are all Double Ring all-reduces.

Table 6. Different results of RSGCM under different LBR

		LBR	
		10:1	1:10
N	9	$ring^{2,1}_{3,6} + ring^{2,1}_{3,6}$	$ring^{2,1}_{8,2}$
	16	$ring^{2,2}_{6,4} + ring^{2,2}_{4,6}$	$ring^{2,1}_{16,2}$
	25	$ring^{2,1}_{25,2}$	$ring^{2,1}_{25,2}$

Here we provide theoretical proof for the contradiction between the intuition and the results in Table 6. We compare the performance of 2D-Mesh[68] and Double Ring all-reduce under an $\sqrt{N} * \sqrt{N}$ matrix.

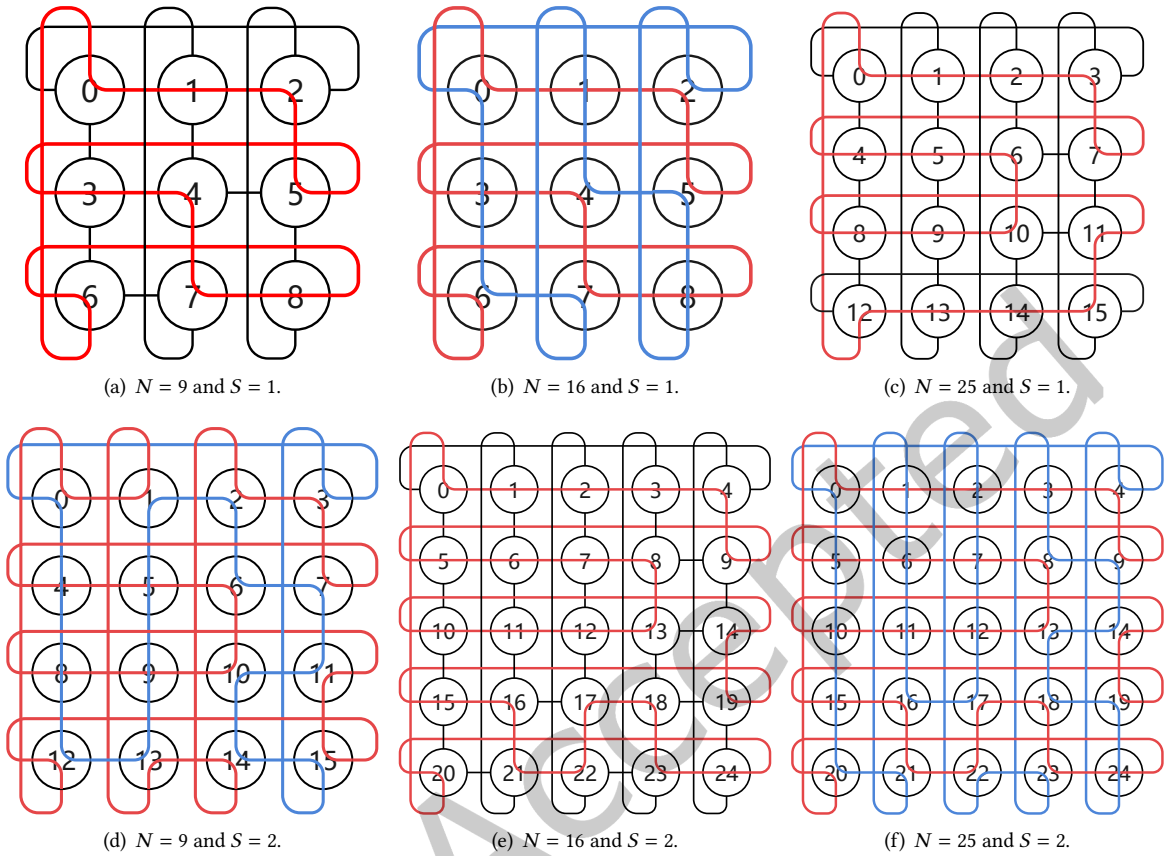
- For 2D-Mesh[68], from Table 4 we can get its performance model shown in (16)

$$T_m(D,N) = 2T_1(\frac{D}{2}, \max(\sqrt{N},\sqrt{N})) - 4T_3(\frac{D}{2}) = 2 * (2\tau_L\sqrt{N} - \frac{2\tau_B\frac{D}{2}}{\sqrt{N}}) - 4(\tau_L - \tau_B\frac{D}{2}) \tag{16}$$

- For Double Ring all-reduce, from Table 4 we can also get its performance model shown in (17).

$$T_r(D,N) = T_1(\frac{D}{2}, N) - 2T_3(\frac{D}{2}) = 2\tau_L N - \frac{2\tau_B\frac{D}{2}}{N} - 2(\tau_L - \tau_B\frac{D}{2}) \tag{17}$$

If we are looking for the LBR, which has better performance with 2D-Mesh[68] than Double Ring all-reduce, (18) should be satisfied.



(a) $N = 9$ and $S = 1$. (b) $N = 16$ and $S = 1$. (c) $N = 25$ and $S = 1$.

(d) $N = 9$ and $S = 2$. (e) $N = 16$ and $S = 2$. (f) $N = 25$ and $S = 2$.

Fig. 8. The training result for the 2-dimension mesh grid connection matrix for different N and S. Red lines indicate the rings in the first ring-set, and the blue lines indicate the rings in the second ring-set.

$$T_r(D,N) > T_m(D,N) \Rightarrow LBR > \frac{D}{2N} \tag{18}$$

According to the result of (18), we find a performance boundary which has different optimal solution on different sides. When using $3 * 3$ connection matrix and data size of $D = 32MB$, the performance boundary is $LBR = \frac{16}{9}MB$. Thus for the results in Fig. 8 and Table 6, we can derive that when $LBR = \frac{0.009}{0.039}MB = \frac{9}{39}MB$ and when $LBR = \frac{0.01}{0.1}MB = \frac{1}{10}MB$, Double Ring all-reduce has better performance and when $LBR = \frac{0.1}{0.01}MB = \frac{10}{1}MB$, 2D-Mesh[68] has better performance.

In addition, when there are 16 or 25 devices, although from (18) we can derive that the performance boundary is $LBR = 1MB$ and $LBR = \frac{16}{25}MB$, it appears that Fig. 8 and Table 6 don't follow the optimal solution to choose 2D-Mesh[68]. The reason is the trade-off between the training time and the solution's performance.

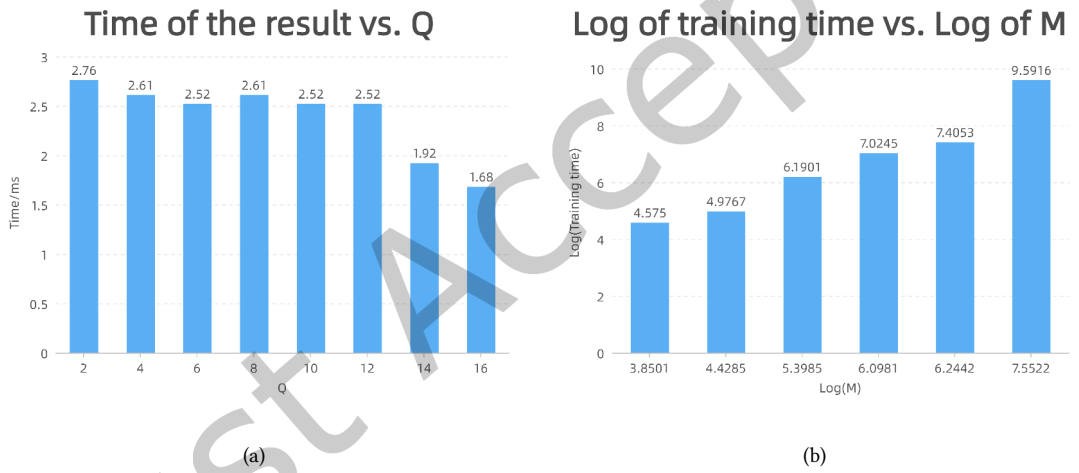
The third is the value of Ep. Under different values of Ep, we perform RSGCM on the $4 * 4$ connection matrix and get the result in Fig. 9(a). In the figure, simulation parameter $Q = \frac{Ep}{M}$. Half of the episodes adapt exploration during training, and the other half samples actions based on experience.

From Fig. 9(a), we can observe that the agent tends to find a better communication schedule with more training episodes.

Fourth, we estimate the relationship between the time used for simulation and the size of the searching space. It appears that from Table 7 where the number of connections is denoted as C, the time used for the construction of the Searching Space t_C is not relative to any of the parameters. Also, the simulation time t_S and the size of the Searching Space M is close to a linear relationship shown in Fig. 9(b). The result is a direct outcome since the number of simulation episodes is 8 times the size of the Searching Space.

Table 7. The construction time of the Searching Space and the searching time for the communication scheduling algorithm under different N and S

N	C	S	M	t_C/s	t_S/s	N	C	S	M	t_C/s	t_S/s	N	C	S	M	t_C/s	t_S/s
9	18	1	47	0.27	97	16	32	1	70	140.15	145	25	50	1	221	31.58	488
		2	445	0.45	1124			2	515	281.26	1363			2	1905	36.07	14641



(a) (b)

Fig. 9. (a) The relationship between the simulation episodes, represented by simulation parameter Q, and the performance of the final communication schedule. (b) The logarithm histogram of the relationship between the searching space size and the searching time.

5 RELATED WORK

Optimization of network collaboration. [8] optimally partitions shared resources among various applications over a regular edge infrastructure to guarantee a convergence bound and an optimal resource assignment. [40] formulates a flow-based delay cost minimization problem and uses entropic surjectivity to measure the sparsity of the function and the limits of computation. [72] is a series of cost-efficient strategies that exploit cheaper volatile cloud instances based on quantifying the relationship between active worker nodes, SGD convergence, and training time. [70] defines cut metrics that measure the connectivity and identify the nonzero gap between

the maximum flow and the minimum cut. It develops polynomial-time approximation algorithms to compute the optimal interdiction for a network flow interdiction problem. [64] is a network-aware distributed learning optimization methodology that trade off costs based on devices processing, offloading, and discarding data points, which improves network resource utilization. [39] uses a lightweight trapdoor compression method for data encryption on mobile to optimize communication efficiency.

Optimization of network robustness. [24] is a recursively defined data center structure called DCell, which has higher network capacity and is fault-tolerant since it does not have a single point of failure and its distributed fault-tolerant routing protocol performs near shortest-path routing. [23] is a network architecture called BCube and designed explicitly for shipping-container-based, modular data centers, whose server-centric network structure can speed up one-to-one, one-to-several, and one-to-all traffic patterns. [12] uses edge rewiring methods to enhance the loops in the network and results in the improvement of connectivity robustness.[67] uses dependency link centrality product index to identify key fragile dependency links and removes them to enhance the robustness of coupled networks. [76] is a per-packet transmission scheme that reduces the impact of packet reordering and deals with uncertainties in asymmetric networks while using a coding technique to reduce long-tailed flow completion time under network asymmetry. [47] uses cloud live forensics to acquire and preserve the reliable data in cloud computing systems by reducing Trusted Computing Base (TCB) size, collecting evidence directly from the hardware, and protecting the evidence and other sensitive files with Filesafe module. [57] uses an optimization method whose individual node degrees are balanced iteratively based on the No-Regret learning algorithm to increase the resilience against outside attacks for network topology.[2] uses an equivalent multilevel programming approach for networks whose nodes with nonzero demand or supply are relatively sparse, based on the recursive application of the equivalent bilevel formulation to facilitate an easy solution to the multilevel programming formulation for tree-reducible networks.

Optimization for DML. [5] is an online polynomial-time algorithm for scheduling the arriving DML jobs and deciding the adjusted numbers of concurrent workers and parameter servers for each job to maximize the overall utility of all jobs. [62] is an asynchronous distributed machine learning framework called SIREN that uses a swarm of stateless functions, controlled by a deep reinforcement learning scheduler, to achieve higher parallelism, higher elasticity, and lower system configuration overhead. [4] is a communication scheduler called PACE containing the best tensor-preemptive communication schedule identifying and scheduling oracle that preemptively schedules all-reduce tensors based on the DAG of DNN training, which guarantees maximal communication overlapping over computation and high bandwidth utilization. [36] employs momentum correction, local gradient clipping, momentum factor masking, and warm-up training to reduce the communication bandwidth while preserving accuracy. [63] is a gradient aggregation algorithm that runs on BCube, instead of Fat-Tree topology to achieve higher network performance and lower network cost. [61] is a scheduling system called Blink that uses spanning trees to find the optimal communication schedules based on tree structures that improve the efficiency of the communication schedules, which can also be used to increase the robustness of gradient aggregation. [43] focuses on the benefits of co-design of all-reduce algorithm and the network system. [25] focuses on a specific kind of DML, which is called Federated Learning(FL). It uses an alarming proactive technique to mitigate the effect of malicious clients to provide a convergence guarantee to the DML tasks.

Optimization of all-reduce. [11] is a communication library for DML called BlueConnect that decomposes a single all-reduce operation into a large number of parallelizable reduce-scatter and all-gather operations to exploit the trade-off between latency and bandwidth and adapt to a variety of network configurations. [32] is an efficient and flexible all-reduce algorithm called FlexReduce for DML under asymmetric or irregular network hierarchies. It deals with the heterogeneity of devices in a distributed system to mitigate the communication overhead. [6] encodes its synthesis as a quantifier-free SMT formula and solves the formula with Z3-solver to find latency-optimal or bandwidth-optimal collectives. [9] analyzes the performance of MPI all-reduce and accelerate the process with pipelined approach. [59] provides hierarchical all-reduce algorithm with pipelined approach,

which is similar to 2DHRA[29]. [30] uses reduce_scatter and allgatherv to perform the all-reduce process, which is similar to 2D-Torus[41]. [38] proposes a scalable fully-pipelined architecture that handles tasks like forwarding, aggregation, and retransmission with no bandwidth loss. [16] focuses on custom operators and data types, with sparse data and reproducible aggregation.

Optimization algorithms. For traditional optimization algorithms, there is enumeration algorithm[14]. [14] samples all possible solutions from the solution space and finds the optimal one. It guarantees to find the optimal solution but is computationally expensive. [7] uses solvers for the problem but need careful formulation of the scenario. It can obtain the optimal solution but takes a long time to find an available solution. For heuristic algorithms, there are a large number of algorithms, including greedy algorithm[14], Hill Climbing(HC)[51], Simulated Annealing(SA)[18], Particle Swarm Optimization(PSO)[13], and Ant Colony Optimization(ACO)[17]. [14] simply chooses the best action according to the current state, ignoring the previous action, so it does not guarantee to find the global optimal but has the fastest speed. [51] originated from the greedy algorithm, but it chooses the best solution according to a small range of possible solutions instead of the whole solution space. [18] is an extension of Monte Carlo sampling. It has a high exploration rate at the beginning, and optimize the solution by searching in its neighbors. [13] simulates the foraging action of birds. It initializes several particles that have their location and velocity. The particles move towards the optimal solution with a better fitness score according to the information from all particles. [17] simulates the action of ants. It keeps track of the trajectories of each ant and collects the quality of the trajectories to update the knowledge of all ants, which guides other ants to follow a better trajectory. For AI optimization approaches, reinforcement learning algorithms, including REINFORCE[66], DDPG[35], A3C[42], and PPO[50], are good choices. [66] is a basic Monte Carlo-based policy gradient algorithm that samples an episode and updates the action possibility. [35] computes the action directly from the strategy function, which deals with continuous action space. It is a deterministic strategy that is computationally cheaper. [42] is based on the actor-critic method and experiences replay and multi-threading techniques to achieve higher training speed and better convergence. [50] changes the original policy gradient algorithm to an off-policy approach and introduces a penalty function to maintain the convergence quality.

First, for tasks, our approach focuses on a single gradient aggregation task instead of a group of jobs[5, 8], computation time on each node[40], asymmetric networks[11, 32, 76], multilevel programming formulation for tree-reducible networks[2], or network system co-design[43]. Second, for designing goal, RSGCM aims to increase the robustness of gradient aggregation instead of other metrics like efficiency[6, 9, 30, 39, 59, 62], bandwidth loss[38], connectivity[12, 23, 24, 67, 70], resilience of attacks on connectivity[57], convergence protection[25], resource utilization[64], or cloud reliability[47]. Third, for input, our approach optimizes the communication schedules based on a given physical topology, but not the tensor size[4, 36] or the number of active worker node[72]. Fourth, for structures, our approach is based on ring-based all-reduce algorithms instead of tree-based algorithms[61, 63].

6 CONCLUSION

In this paper, we have proposed Robust Searching-based Gradient Collaborative Management(RSGCM) in Intelligent Transportation System, a gradient managing algorithm that can increase the robustness of the gradient aggregation for MCC in ITS. It finds the near-optimal communication schedule to be performed on a given connection matrix with a relatively acceptable time cost and acceptable performance. We use Monte Carlo Tree Search to decide what action to take when facing certain data distributions. The environment can evaluate the choice from the Monte Carlo Tree and update the decision-making agent. The experiments show that RSGCM can restore the efficiency of the gradient aggregation within several training episodes and represent different scenarios through the modification of parameters. This algorithm has an identical value in a practical environment. We

have the prediction that this kind of gradient managing algorithm might have its usage in the industrial field of ITS.

ACKNOWLEDGMENTS

This work was supported in part by National NSF of China (NO. 61872234, 61732010), Shanghai Key Laboratory of Scalable Computing and Systems, Innovative Research Foundation of Ship General Performance (NO.25622114), SJTU Library-Jiangsu Jiatu Future Library Smart Service Joint R&D Center and the Key Laboratory of PK System Technologies Research of Hainan.

REFERENCES

[1] Samah Aloufi and Abdulmotaleb El Saddik. 2022. MMSUM digital twins: A multi-view multi-modality summarization framework for sporting events. *ACM Transactions on Multimedia Computing, Communications, and Applications* 18, 1 (2022), 1–25.
[2] Qin Ba and Ketan Savla. 2017. Robustness of DC networks with controllable link weights. *IEEE Transactions on Control of Network Systems* 5, 3 (2017), 1479–1491.
[3] Baidu. 2016. *Baidu-allreduce.* https://github.com/baidu-research/baidu-allreduce
[4] Yixin Bao, Yanghua Peng, Yangrui Chen, and Chuan Wu. 2020. Preemptive all-reduce scheduling for expediting distributed DNN training. In *Proceedings of IEEE INFOCOM 2020 IEEE Conference on Computer Communications*. 626–635.
[5] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. 2018. Online job scheduling in distributed machine learning clusters. In *Proceedings of IEEE INFOCOM 2018 IEEE Conference on Computer Communications*. 495–503.
[6] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, et al. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 62–75.
[7] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, et al. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 62–75.
[8] Gabriele Castellano, Flavio Esposito, and Fulvio Risso. 2019. A distributed orchestration algorithm for edge computing resources with guarantees. In *Proceedings of IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. 2548–2556.
[9] Adrián Castelló, Enrique S Quintana-Ortí, and José Duato. 2021. Accelerating distributed deep neural network training with pipelined MPI allreduce. *Cluster Computing* 24, 4 (2021), 3797–3813.
[10] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).
[11] Minsik Cho, Ulrich Finkler, Mauricio Serrano, David Kung, and Hillery Hunter. 2019. BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. *IBM Journal of Research and Development* 63, 6 (2019), 1–1.
[12] Masaki Chujyo and Yukio Hayashi. 2021. A loop enhancement strategy for network robustness. *Applied Network Science* 6, 1 (2021), 1–13.
[13] Maurice Clerc. 2010. *Particle swarm optimization*. Vol. 93.
[14] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*.
[15] Jacob Devlin, Mingwei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1. 4171–4186.
[16] Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, et al. 2020. PsPIN: A high-performance low-power architecture for flexible in-network compute. *arXiv preprint arXiv:2010.03536* (2020).
[17] Marco Dorigo. 2007. Ant colony optimization. *Scholarpedia* 2, 3 (2007), 1461.
[18] Kathryn Anne Dowsland and Jonathan Thompson. 2012. Simulated annealing. *Handbook of natural computing* (2012), 1623–1655.
[19] Facebook. 2019. *PyTorch.* https://pytorch.org/docs/stable/index.html
[20] Message Passing Interface Forum. 2015. *MPI: A Message-Passing Interface Standard Version 3.1.* https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
[21] Zan Gao, Yinming Li, and Shaohua Wan. 2020. Exploring deep learning for view-based 3D model retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications* 16, 1 (2020), 1–21.
[22] Google. 2021. *TensorFlow.* https://tensorflow.google.cn/guide
[23] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, et al. 2009. BCube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. 63–74.
[24] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. 2008. Dcell: A scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. 75–86.

[25] Hanxi Guo, Hao Wang, Tao Song, Yang Hua, Zhangcheng Lv, Xiulang Jin, et al. 2021. Siren: Byzantine-robust federated learning via proactive alarming. In *Proceedings of the ACM Symposium on Cloud Computing*. 47––60.
[26] Roger W Hockney. 1994. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.* 20, 3 (1994), 389–398.
[27] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.
[28] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, et al. 2018. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205* (2018).
[29] Youhe Jiang, Huaxi Gu, Yunfeng Lu, and Xiaoshan Yu. 2020. 2D-HRA: Two-dimensional hierarchical ring-based all-reduce algorithm in large-scale distributed machine learning. *IEEE Access* 8 (2020), 183488–183494.
[30] Andreas Jocksch, Noé Ohana, Emmanuel Lanti, Eirini Koutsaniti, Vasileios Karakasis, and Laurent Villard. 2021. An optimisation of allreduce communication in message-passing systems. *Parallel Comput.* 107 (2021), 102812.
[31] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.
[32] Jinho Lee, Inseok Hwang, Soham Shah, and Minsik Cho. 2020. FlexReduce: Flexible all-reduce for distributed deep learning on asymmetric network topology. In *Proceedings of 2020 57th ACM/IEEE Design Automation Conference*. 1–6.
[33] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, et al. 2019. Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUdirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.
[34] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, et al. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*. 583–598.
[35] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, et al. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
[36] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).
[37] Yangxin Lin, Ping Wang, and Meng Ma. 2017. Intelligent transportation system (ITS): Concept, challenge and opportunity. In *Proceedings of 2017 IEEE 3rd International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security*. 167–172.
[38] Yao Liu, Junyi Zhang, Shuo Liu, Qiaoling Wang, Wangchen Dai, and Ray Chak Chung Cheung. 2021. Scalable fully pipelined hardware architecture for in-network aggregated AllReduce communication. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 10 (2021), 4194–4206.
[39] Ruhui Ma, Jian Li, Haibing Guan, Mingyuan Xia, and Xue Liu. 2015. EnDAS: Efficient encrypted data search as a mobile cloud service. *IEEE Transactions on Emerging Topics in Computing* 3, 3 (2015), 372–383.
[40] Derya Malak, Alejandro Cohen, and Muriel Médard. 2020. How to distribute computation in networks. In *Proceedings of IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. 327–336.
[41] Hiroaki Mikami, Hisahiro Suganuma, Yoshiki Tanaka, Yuichi Kageyama, et al. 2018. Massively distributed SGD: ImageNet/ResNet-50 training in a flash. *arXiv preprint arXiv:1811.05233* (2018).
[42] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, et al. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.
[43] Truong Thao Nguyen and Mohamed Wahib. 2021. An Allreduce algorithm and network co-design for large-scale training of distributed deep learning. In *Proceedings of 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing*. 396–405.
[44] NVIDIA. 2020. *NVIDIA Collective Communication Library (NCCL) Documentation.* https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html
[45] NVIDIA. 2022. *CUDA Toolkit Documentation.* https://docs.nvidia.com/cuda/index.html
[46] Luo Qi. 2008. Research on intelligent transportation system technologies and applications. In *Proceedings of 2008 Workshop on Power Electronics and Intelligent Transportation System*. 529–531.
[47] Zhengwei Qi, Chengcheng Xiang, Ruhui Ma, Jian Li, Haibing Guan, and David S. L. Wei. 2017. ForenVisor: A tool for acquiring and preserving reliable Data in Cloud Live Forensics. *IEEE Transactions on Cloud Computing* 5, 3 (2017), 443–456.
[48] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.
[49] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.
[50] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[51] Bart Selman and Carla P Gomes. 2006. Hill-climbing search. *Encyclopedia of cognitive science* 81 (2006), 82.

[52] Alexander Sergeev and Mike Del Balso. 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).
[53] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
[54] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.
[55] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354–359.
[56] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations*.
[57] Insoo Sohn. 2019. Robustness enhancement of complex networks via No-Regret learning. *ICT Express* 5, 3 (2019), 163–166.
[58] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.
[59] Truong Thao Nguyen, Mohamed Wahib, and Ryousei Takano. 2021. Efficient MPI-AllReduce for large-scale deep learning on GPU-clusters. *Concurrency and Computation: Practice and Experience* 33, 12 (2021), 5574.
[60] Shaohua Wan, Zan Gao, Hanwang Zhang, and Xiaojun Chang. 2021. Introduction to the special issue on fine-grained visual computing. , 3 pages.
[61] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. 2020. Blink: Fast and generic collectives for distributed ML. *Machine Learning and Systems* 2 (2020), 172–186.
[62] Hao Wang, Di Niu, and Baochun Li. 2019. Distributed machine learning with a serverless architecture. In *Proceedings of IEEE INFOCOM 2019 IEEE Conference on Computer Communications*. 1288–1296.
[63] Songtao Wang, Dan Li, Yang Cheng, Jinkun Geng, Yanshu Wang, Shuai Wang, et al. 2018. BML: A high-performance, low-cost gradient synchronization algorithm for dml training. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 4243–4253.
[64] Su Wang, Yichen Ruan, Yuwei Tu, Satyavrat Wagle, Christopher G Brinton, and Carlee Joe-Wong. 2021. Network-aware optimization of distributed learning for fog computing. *IEEE/ACM Transactions on Networking* (2021).
[65] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. 2016. Ako: Decentralised deep learning with partial gradient exchange. In *Proceedings of the 7th ACM Symposium on Cloud Computing*. 84–97.
[66] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.
[67] Xuhua Yang, Wenhao Feng, Guang Chen, Lei Wang, Tao Zou, and Peng Jiang. 2020. Enhancing coupled networks robustness via removing key fragile dependency links. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 3 (2020), 953–957.
[68] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. 2018. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992* (2018).
[69] Jiaru Zhang, Yang Hua, Tao Song, Hao Wang, Zhengui Xue, Ruhui Ma, et al. 2022. Improving bayesian neural networks by adversarial sampling. (2022).
[70] Jianan Zhang, Hyang-Won Lee, and Eytan Modiano. 2019. On the robustness of distributed computing networks. In *Proceedings of 2019 15th International Conference on the Design of Reliable Communication Networks*. 122–129.
[71] Wei Zhang, Ting Yao, Shiai Zhu, and Abdulmotaleb El Saddik. 2019. Deep learning–based multimedia analytics: a review. *ACM Transactions on Multimedia Computing, Communications, and Applications* 15, 1s (2019), 1–26.
[72] Xiaoxi Zhang, Jianyu Wang, Gauri Joshi, and Carlee Joe-Wong. 2020. Machine learning on volatile instances. In *Proceedings of IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. 139–148.
[73] Yin Zhang, Xiao Ma, Jing Zhang, M Shamim Hossain, Ghulam Muhammad, and Syed Umar Amin. 2019. Edge intelligence in the cognitive Internet of Things: Improving sensitivity and interactivity. *IEEE Network* 33, 3 (2019), 58–64.
[74] Huasha Zhao and John Canny. 2013. Butterfly mixing: Accelerating incremental-update algorithms on clusters. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. 785–793.
[75] Yi Zheng, Yong Zhou, Jiaqi Zhao, Ying Chen, Rui Yao, Bing Liu, and Abdulmotaleb El Saddik. 2022. Clustering matters: Sphere feature for fully unsupervised person re-identification. *ACM Transactions on Multimedia Computing, Communications, and Applications* 18, 4 (2022), 1–18.
[76] Shaojun Zou, Jiawei Huang, Jianxin Wang, and Tian He. 2019. Improving TCP robustness over asymmetry with reordering marking and coding in data centers. In *Proceedings of 2019 IEEE 39th International Conference on Distributed Computing Systems*. 57–67.