RESEARCH-ARTICLE

SCALE-CCL: A Scalable Collective Communication Library for Wide-Area Distributed Training

JIAHENG XIONG, Politecnico di Milano, Milan, MI, Italy

QIAOLUN ZHANG, Politecnico di Milano, Milan, MI, Italy

PAOLO MEDAGLIANI, Huawei Technologies Co., Ltd., Shenzhen, Guangdong, China

MICHELE FERRERO, Huawei Technologies Co., Ltd., Shenzhen, Guangdong, China

XIAOMIN LIU, Politecnico di Milano, Milan, MI, Italy

MENG LIAN, Politecnico di Milano, Milan, MI, Italy

View all

Open Access Support provided by:

Huawei Technologies Co., Ltd.

Politecnico di Milano

Published: 30 November 2025

Citation in BibTeX format

CoNEXT '25:The 21st International Conference on emerging Networking EXperiments and Technologies
December 1 - 4, 2025
Hong Kong, Hong Kong

Conference Sponsors:
SIGCOMM

INET4AI '25: Proceedings of the 1st Workshop on Inter-networking challenges for AI (December 2025)
https://doi.org/10.1145/3769695.3771677
ISBN: 9798400722417

SCALE-CCL: A Scalable Collective Communication Library for Wide-Area Distributed Training

Jiaheng Xiong
Politecnico di Milano
Milan, Italy
jiaheng.xiong@polimi.it

Qiaolun Zhang*
Politecnico di Milano
Milan, Italy
qiaolun.zhang@polimi.it

Paolo Medagliani
Huawei Technologies Ltd.
Paris, France
paolo.medagliani@huawei.com

Michele Ferrero
Huawei Technologies Ltd.
Paris, France
ferrero.michele@huawei.com

Xiaomin Liu
Politecnico di Milano
Milan, Italy
xiaomin.liu@mail.polimi.it

Meng Lian
Politecnico di Milano
Milan, Italy
meng.lian@mail.polimi.it

Nicola Di Cicco
Politecnico di Milano
Milan, Italy
nicola.dicicco@polimi.it

Baosen Zhao
Politecnico di Milano
Milan, Italy
baosen.zhao@mail.polimi.it

Mëmëdhe Ibrahimi
Politecnico di Milano
Milan, Italy
memedhe.ibrahimi@polimi.it

Francesco Musumeci
Politecnico di Milano
Milan, Italy
francesco.musumeci@polimi.it

Massimo Tornatore
Politecnico di Milano
Milan, Italy
massimo.tornatore@polimi.it

Abstract

Collective Communication Libraries (CCLs) are widely used to coordinate and optimize data exchange among multiple GPUs. As training clusters increasingly span multiple datacenters for scalability and resource pooling, applying CCLs over wide-area networks (WANs) becomes essential. However, existing CCLs are primarily designed for stable, low-latency intra-datacenter environments. In contrast, WANs exhibit dynamic and unpredictable conditions that limit the effectiveness of traditional CCLs. While recent solutions adopt global optimization techniques, such as Integer Linear Programming (ILP), to improve communication efficiency, these methods assume static topologies and full network visibility, which are often unrealistic in WAN settings with link variability, limited observability, and dynamic traffic patterns. To address these challenges, we propose **SCALE-CCL**, a scalable heuristic algorithm for optimizing AllGather operations in WAN environments. SCALE-CCL derives global schedules offline from lightweight aggregated metrics (e.g., sampled bandwidth, queue status, and reception history). Thanks to its sub-second synthesis time, schedules can be recomputed whenever needed, either before each training round or when significant WAN changes occur, without noticeable overhead. Compared to the state-of-the-art ILP-based solution TE-CCL and baselines including a shortest-path heuristic (SPH) and NCCL, SCALE-CCL reduces scheduling time by up to four orders of magnitude, while achieving AllGather completion times within a 10% gap of TE-CCL in more than 90% of cases (never exceeding 15%). It consistently outperforms SPH and generally surpasses NCCL in WAN environments.

CCS Concepts

• **Networks** → **Network design and planning algorithms**; • **Computer systems organization** → *Interconnection architectures.*

Keywords

Collective Communication, Distributed Machine Learning, Topology-Aware Scheduling, GPU Cluster Optimization

ACM Reference Format:
Jiaheng Xiong, Qiaolun Zhang, Paolo Medagliani, Michele Ferrero, Xiaomin Liu, Meng Lian, Nicola Di Cicco, Baosen Zhao, Mëmëdhe Ibrahimi, Francesco Musumeci, and Massimo Tornatore. 2025. SCALE-CCL: A Scalable Collective Communication Library for Wide-Area Distributed Training. In *Proceedings of the 1st Workshop on Inter-networking Challenges for AI (INET4AI '25), December 1–4, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3769695.3771677

1 Introduction

As machine learning models, and especially large language models (LLMs), grow in size and complexity, efficiently training them increasingly requires distributing computation across multiple GPUs. Initially, this distribution takes place within a single server or datacenter (intra-DC), where high-bandwidth and low-latency interconnects enable efficient collective communication. However, as models and datasets continue to grow, training clusters often need

*Corresponding author.

INET4AI '25, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-2241-7/2025/12
https://doi.org/10.1145/3769695.3771677

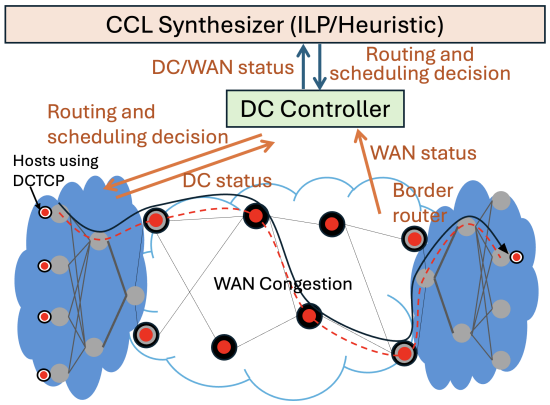

Figure 1: System Model for CCL over Cross Region DC

to span multiple datacenters in different geographic regions (inter-DC) [13, 14]. Beyond scale, factors such as data privacy and regulatory constraints, data locality, and resource scarcity within a single DC further motivate inter-DC training [5, 6]. These geo-distributed deployments rely on wide-area networks (WANs) to interconnect compute resources across DCs, introducing communication delays, bandwidth limitations, and diverse link characteristics that differ substantially from those in intra-DC environments.

Collective communication primitives such as *AllGather*, *AllReduce*, and *AlltoAll* are fundamental operations coordinating data exchange among multiple GPUs or accelerators during distributed training. They are typically implemented and optimized by *Collective Communication Libraries (CCLs)* such as MPI [7], NCCL [18], or TACCL [19], which enable scalable training across modern GPU-based distributed infrastructures. Prior work has made significant progress in optimizing routing and scheduling decisions of CCLs [15, 19]. A representative example is *TE-CCL* [15], which formulates collective communication as a multi-commodity flow optimization and solves it via Integer Linear Programming (ILP). While these methods deliver near-optimal performance, they were designed for *intra-DC networks*, assuming static link capacity and requiring long computation times (thousands of seconds even for topologies with a few tens of GPUs).

Among collective communication primitives, *AllGather* is particularly critical and challenging in WAN settings. AllGather is an operation where each GPU starts with a distinct chunk of data and ends up collecting all chunks from all participating GPUs. In WAN environments the main challenge arises from unpredictable delay fluctuations, which make it difficult to design schedules that remain efficient across changing conditions [21]. We therefore focus on developing an AllGather algorithm for WAN environments that combines low scheduling overhead with high performance.

To this end, we propose **SCALE-CCL**, a Scalable Collective Communication Library for wide-area distributed training. SCALE-CCL employs a lightweight heuristic that operates on aggregated link statistics and performs schedule computation centrally within an offline synthesizer. Thanks to its sub-second synthesis time, SCALE-CCL can be re-executed whenever network conditions change, providing a favorable trade-off between responsiveness and solution quality for WAN-scale AI deployments.

This paper provides the following contributions:

- We define, for the first time to the best of our knowledge, the *WAN-aware CCL scheduling problem* for cross-DC AI training, and use *AllGather* as a representative case for numerical analysis.
- We propose SCALE-CCL, a centralized offline heuristic that computes globally coordinated schedules from lightweight aggregated link statistics, enabling scalability and rapid re-execution under dynamic WAN conditions.
- We demonstrate that SCALE-CCL achieves more than 1000× faster scheduling than TE-CCL, with at most 15% longer AllGather completion time.

The remainder of the paper is organized as follows. Section 2 reviews related work on collective communication and WAN-aware optimization. Section 3 defines the system model and WAN-aware CCL scheduling problem. Section 4 describes our scheduling algorithm using aggregated link statistics. Section 5 presents numerical results, comparing SCALE-CCL with TE-CCL and shortest-path heuristics under different WAN delays, chunk sizes, and network connectivity. Section 6 concludes and outlines future directions.

2 Related Work

Collective communication is a key component in distributed AI training and has been extensively studied in intra-DC environments, whereas inter-DC collectives have received far less attention. Traditional libraries such as MPI [8], NCCL [18], RCCL [2], and oneCCL [10] implement common operations (e.g., AlltoAll, AllGather) using static strategies such as rings or trees [22]. These methods are highly optimized for homogeneous, low-latency settings, assuming uniform bandwidth and stable topologies, assumptions that break down in modern deployments with diverse link capacities, heterogeneous delays, and dynamic congestion. We therefore also include NCCL as a widely deployed intra-DC baseline in our evaluation (Section 5), to gauge its behavior over WAN links without WAN-specific tuning.

To improve efficiency in intra-DC clusters, several works propose topology-aware scheduling. SCCL [3] and TACCL [19] synthesize collective algorithms (via SMT or communication sketches), achieving performance comparable to hand-tuned implementations. However, this comes at the cost of very high synthesis times, which can extend to hours and, for larger topologies, even exceed 24 hours, making them impractical for dynamic or large-scale deployments.

Optimizing collectives across multiple datacenters interconnected by WANs remains an open challenge. WAN-aware traffic engineering systems such as B4 [11] and SWAN [20] focus on point-to-point or multicast flows rather than many-to-many synchronization. Recent works on multi-DC training [4, 17] largely rely on coarse-grained or static strategies that cannot adapt to fluctuating WAN conditions. In particular, their scalability is limited with respect to the number of chunks that must be exchanged, which directly affects completion time in inter-DC collectives. To the best of our knowledge, no prior work has investigated collective communication libraries specifically designed for multi-DC environments.

3 System Model and Problem Statement

We model an intra-DC network as an undirected graph $G(V, E)$, where V denotes GPUs and border routers (BRs), and E the set of logical links. To capture inter-DC communication, we abstract the WAN as direct logical connections between BRs, representing

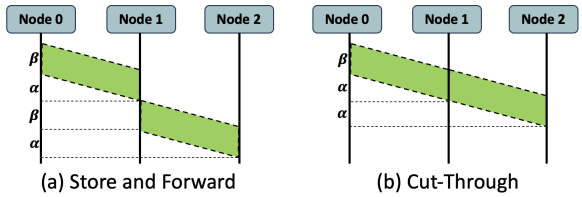

Figure 2: Illustration of forwarding models

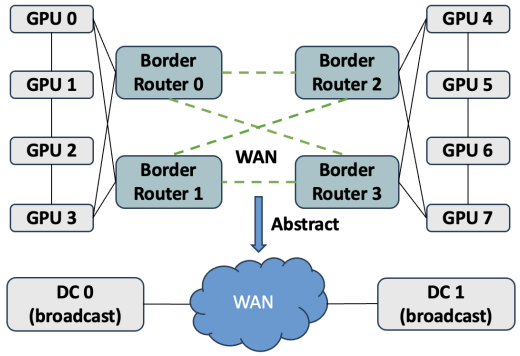

Figure 3: Logical Topology

dominant inter-DC paths inferred from routing policies or traffic engineering systems, as in B4 [11] and SWAN [20]. We consider distributed training workloads where each GPU initially holds one *chunk* of size L, representing its local portion of data to be exchanged. A chunk may be further divided into smaller *subchunks* when finer-grained scheduling is required. During `AllGather`, every GPU must collect $n-1$ chunks held by other GPUs, potentially via multi-hop paths across GPUs and BRs. This makes `AllGather` a representative many-to-many collective primitive widely used in distributed training [18, 21]. Each link $e \in E$ is characterized by propagation delay α_e and bandwidth b_e. Transmitting a chunk of size L over link e incurs transmission delay $\beta_e = \frac{L}{b_e}$.

We model GPU nodes with *store-and-forward* forwarding (as in Ref. [15]): a node must receive an entire subchunk before forwarding. For a path $P = (e_1, \ldots, e_h)$ with per-link propagation delays α_{e_i} and bandwidths b_{e_i}, transmitting a subchunk of size L_s over link e takes time $\beta_e(L_s) = \frac{L_s}{b_e}$. The end-to-end completion time under store-and-forward is $T_{\mathrm{SF}}(L_s; P) = \sum_{i=1}^{h} (\alpha_{e_i} + \beta_{e_i}(L_s))$. For homogeneous links ($\alpha_{e_i} = \alpha$, $b_{e_i} = b$), this simplifies to $T_{\mathrm{SF}} = h(\alpha + \beta)$ with $\beta = \frac{L_s}{b}$, as shown in Fig. 2(a). Border routers (BRs) use *cut-through forwarding*: forwarding starts once data is received [12]. Since WAN packets are typically small (e.g., < 1.5 kB) [11], a subchunk is effectively sent as a packet stream pipelined across hops; we therefore approximate packet-level cut-through at the *subchunk* granularity. With sufficient buffering to absorb rate mismatches, completion time along P is $T_{\mathrm{CT}}(L_s; P) = \sum_{i=1}^{h} \alpha_{e_i} + \max_{1 \le i \le h} \beta_{e_i}(L_s)$, i.e., propagation delays add while the transmission term is limited by the bottleneck link. In the homogeneous case, this reduces to $T_{\mathrm{CT}} = h\alpha + \beta$ with $\beta = \frac{L_s}{b}$, as in Fig. 2(b).

To improve forwarding parallelism, we allow partitioning each chunk into c subchunks, so data can be forwarded in smaller units rather than waiting for the entire chunk. After this division, each GPU must receive $(n-1)c$ subchunks in total. Importantly, only GPU nodes can exploit this mechanism: intermediate GPUs can start forwarding subchunks once they are completely received,

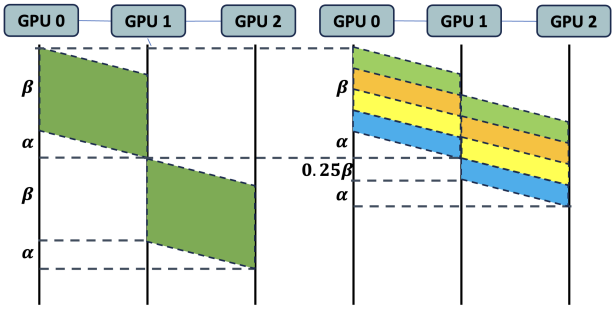

Figure 4: Without Subchunk vs With Subchunk

while BRs are already modeled with cut-through forwarding at the packet level. Over an h-hop GPU-to-GPU path, the delay without subchunking is $T_{\text{no-sub}} = h \times (\alpha + \beta)$, while with c subchunks the delay reduces to $T_{\text{subchunk}} \approx \beta + \frac{\beta}{c}(h-1) + h \times \alpha$, assuming synchronized forwarding across hops. Here, α denotes propagation delay and $\beta = \frac{L}{b}$ the transmission delay for a chunk of size L over bandwidth b. This formula is an optimistic lower bound, since in practice subchunking introduces overhead (e.g., headers, scheduling, buffer management), which constrains c; otherwise, as $c \to \infty$, the behavior converges to cut-through forwarding. The practical choice of c depends on system parameters such as MTU, buffer size, and protocol overhead. Figure 4 illustrates this effect: without subchunking, transmission proceeds sequentially, and each hop must wait for a full chunk before forwarding. With subchunking, forwarding at intermediate GPUs overlaps in time, so different subchunks of the same chunk can traverse different hops concurrently, leading to earlier delivery and shorter completion time.

The *WAN-aware CCL optimization problem* is defined as follows. **Given** a physical network topology with per-link propagation and transmission delays, link bandwidths, the collective operation (i.e., the communication semantics and data to be exchanged), and the chosen subchunk size and count, **determine** the routing paths and transmission schedule of all subchunks. The solution must be designed **subject to** three constraints: link capacity must not be oversubscribed, forwarding causality must be respected (a node can only forward data after receiving it), and delivery completeness must be ensured (all nodes must eventually obtain all subchunks). Finally, **the objective is to minimize the completion time**, defined as the time until all GPUs receive all required subchunks.[1]

4 Scalable Collective Communication Algorithm

In this section, we present the algorithmic details of *SCALE-CCL*, including subchunk scheduling and forwarding strategies.

4.1 SCALE-CCL workflow

SCALE-CCL is a **centralized offline synthesizer** at the DC controller that generates routing and scheduling decisions for CCLs. It is built on discrete-event simulation with four event types—transmission start, transmission completion, reception start, and reception completion—illustrated in Fig. 5. By explicitly modeling these events, SCALE-CCL allows different nodes to receive and forward packets concurrently, improving realism and concurrency.

[1] In this work we use `AllGather` as a representative collective, but the problem formulation and heuristic algorithm are generic and apply to other collectives as well.

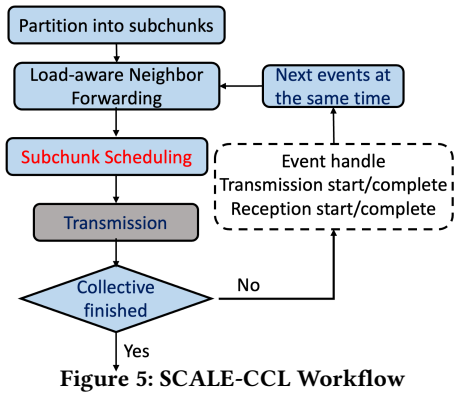

Figure 5: SCALE-CCL Workflow

To tackle inter-DC communication challenges, SCALE-CCL employs distinct scheduling for intra- and inter-DC transfers. Intra-DC networks have stable, fully visible topologies suitable for centralized coordination, while WAN links are dynamic and partially observable, motivating lightweight heuristics. The algorithm inputs (Sec. 3) include: (i) network topology G, (ii) chunk/subchunk parameters (size and number c), (iii) per-link bandwidth and propagation delay, and (iv) the collective operation (AllGather in our case). As shown in Fig. 5, each GPU partitions its local chunk into c equal-sized subchunks. SCALE-CCL operates under a *time- and event-driven* simulator processing all co-timed events *in parallel*. For each subchunk on each link, events include: (i) transmission start, (ii) transmission completion, (iii) reception start, and (iv) reception completion. At each iteration, the synthesizer applies *Load-Aware Neighbor Forwarding (LWNF)* to select next hops, then *Subchunk Scheduling (SS)* to order per-link queues and issue transmissions. The simulator advances to the next timestamp, executes co-timed events, and, if unfinished, handles concurrent next events before looping back to LWNF with updated states. **Crucially, we do not precompute monolithic end-to-end routes**: forwarding is decomposed into *atomic point-to-point events*, and paths *emerge* from executed event sequences while preserving causality (forward only after reception) and link-capacity constraints, without reconstructing the global topology or solving a centralized per-subchunk routing problem.

Scheduling proceeds in two phases. In the **Load-Aware Neighbor Forwarding (LWNF)** phase, each subchunk at node u is assigned to a neighbor $v \in \mathcal{N}(u)$ as $\text{next}(s) = \arg\min_{v \in \mathcal{N}(u)} f(\ell_{uv}, q_v, p_v)$, where ℓ_{uv} is link availability, q_v buffer occupancy, and p_v delivery progress (fraction of received subchunks for that chunk). This balances load and avoids duplication by preferring incomplete neighbors. In the **Subchunk Scheduling (SS)** phase, each node orders pending subchunks to resolve contention and minimize delay. The process iterates event by event until all subchunks reach all nodes.

4.2 Load-Aware Neighbor Forwarding

In the **Load-Aware Neighbor Forwarding (LWNF)** phase, each node decides which neighbor receives each outbound subchunk. This differs from subchunk scheduling, which determines transmission order once forwarding targets are chosen. To support decisions, the system maintains a neighbor memory table recording, for each neighbor, the set of subchunks already received. When forwarding, the algorithm excludes neighbors that already have the subchunk, ensuring only useful transmissions are considered. Among remaining neighbors, the next hop is selected by evaluating link utilization

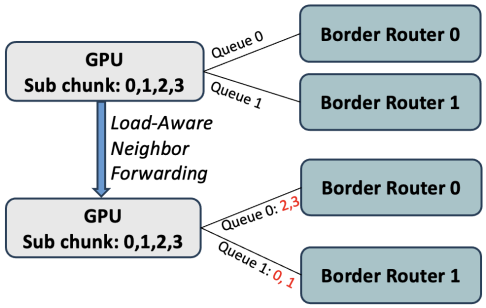

Figure 6: Example of Load-Aware Neighbor Forwarding

and buffer availability. As shown in Fig. 6, a node with four subchunks can split them into multiple queues, each assigned to a different neighbor based on reception states. LWNF applies only to GPU-to-BR and inter-BR queues, since the main goal across the WAN is to quickly forward subchunks to the next DC rather than replicate them at every BR. By contrast, for BR-to-GPU communication, LWNF is relaxed, allowing a BR to broadcast the same subchunk to multiple GPUs. Similarly, intra-DC GPU-to-GPU communication uses direct broadcast, leveraging the high-bandwidth, low-latency topology of GPU clusters. We abstract the WAN as logical BR-to-BR links with estimated propagation delay (Sec. 3), while intra-DC GPUs are assumed interconnected through symmetric high-speed links (e.g., NVLink or Clos/fat-tree) that support efficient local broadcasting. Overall, this strategy avoids redundant transmissions and distributes load across links, improving throughput and reducing completion time. The output of this stage then feeds into the subsequent *Subchunk Scheduling* phase.

4.3 Subchunk Scheduling

Once subchunks are assigned to outgoing queues via Load-Aware Neighbor Forwarding, *SCALE-CCL* performs a lightweight **Subchunk Scheduling (SS)** step to resolve contention and prioritize transmissions. When multiple subchunks are eligible for the same destination, this step selects which to send first, favoring lower estimated delay. The process runs within a centralized offline synthesizer but operates without reconstructing the global network topology. Algorithm 1 outlines the scheduling logic. For each destination d, the algorithm constructs a candidate predecessor set P and a destination set D for contention resolution (Lines 2–3). If d is a BR, P includes all GPUs and peer BRs connected to d, and D is the full set of concurrent receivers for the subchunk. If d is a GPU and the current node u a BR in the same DC, P includes direct predecessors of d and relevant BRs; D includes sibling GPUs in the same DC. Otherwise, P includes only direct predecessors, and $D = \{d\}$.

For each candidate (s, d') pair in $P \times D$, the algorithm examines the respective transmission queue and collects all pending subchunks (Lines 4–6). It estimates the link availability $T[(s, d')]$ as the maximum finish time of previously scheduled transmissions (Line 7). To eliminate redundancy, it identifies duplicate subchunks that appear in multiple queues, temporarily removes them, and stores them in a shared selection buffer (Lines 8–11). Then, for each duplicate subchunk b, it selects the path with the lowest estimated delay and updates the send time accordingly (Lines 12–20). Finally, for the current node u and destination d, the algorithm selects the

Algorithm 1: Subchunk Scheduling

Input: Topology graph G, destination node d, current time t, current node u
Output: Selected subchunk b (or $\emptyset$)

1 Let $\Delta(s,d') = G[s,d'].\texttt{transmission\_delay}$ denote the transmission delay on link (s,d').
2 $P \leftarrow$ predecessor nodes of d and additional sources depending on u;
3 $D \leftarrow$ all candidate destinations that may hold d's data;
4 **foreach** $(s,d') \in P \times D$ **do**
5 **if** (s,d') *has a job queue* **then**
6 $J[(s,d')] \leftarrow$ buffers in $G[s].\texttt{job}[(s,d')]$;
7 $T[(s,d')] \leftarrow \max_{j \in G[s,d'].\texttt{job}} (j.\texttt{sent\_time} - t)$;
8 **foreach** $(s,d'), b \in J$ **do**
9 **if** b *appears in multiple* $J[(s,d')]$ **then**
10 $S[b] \leftarrow S[b] \cup \{(s,d')\}$;
11 remove b in all $J[(s,d')]$;
12 **else**
13 $T[(s,d')] \leftarrow T[(s,d')] + \Delta(s,d')$;
14 **foreach** $b \in S$ **do**
15 **foreach** $(s,d') \in S[b]$ **do**
16 $C[(s,d')] \leftarrow T[(s,d')] + \Delta(s,d')$;
17 $(s^*,d^*) \leftarrow \arg\min_{(s,d')} C[(s,d')]$;
18 $J[(s^*,d^*)] \leftarrow J[(s^*,d^*)] \cup \{b\}$;
19 $T[(s^*,d^*)] \leftarrow T[(s^*,d^*)] + \Delta(s^*,d^*)$;
20 **foreach** $(s,d') \in S[b] \setminus \{(s^*,d^*)\}$ **do**
21 $J[(s,d')] \leftarrow J[(s,d')] \setminus \{b\}$;
22 $Q \leftarrow J[(u,d)]$;
23 **if** $Q = \emptyset$ **then**
24 **return** $\emptyset$;
25 $b \leftarrow Q[0]$;
26 **foreach** $(s,d') \in P \times D, s \neq u$ **do**
27 $J[(s,d')] \leftarrow J[(s,d')] \setminus \{b\}$;
28 **return** b;

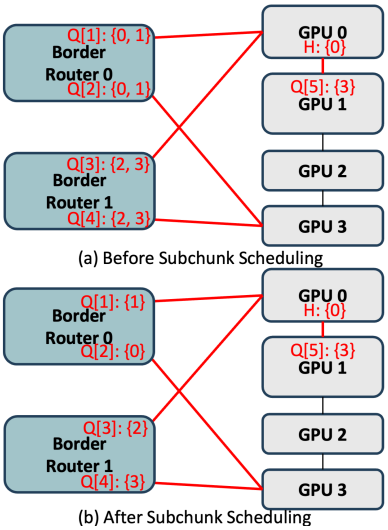

Figure 7: Example of Subchunk scheduling

first subchunk in the scheduled queue and removes it from all other queues to ensure uniqueness of delivery (Lines 21–27).

Fig. 7 illustrates *SCALE-CCL*'s **SS** resolving cross-queue duplication. BR 0 keeps $Q[1]$ and $Q[2]$ (initially $\{0,1\}$); BR 1 keeps $Q[3]$ and $Q[4]$ (initially $\{2,3\}$). GPU 0 already stores subchunk 0 ($H:\{0\}$), and GPU 1 has a pending $Q[5]$ containing 3. Here, $Q[i]$ denotes the scheduling queue toward node i, and H marks subchunks held at a GPU. As shown in Fig. 7(a), red queues participate for destination GPU 0. These are predecessors $P = \{\text{BR0}, \text{BR1}, \text{GPU1}\}$ and destinations $D = \{\text{GPU0}, \text{GPU3}\}$. *Note* $Q[4]$ *is not BR1→GPU0; SS inspects the full Cartesian* $P \times D$ *to resolve duplicates across concurrent receivers (GPU0 and sibling GPU3), not only edges to the destination.* Since GPU 0 already holds subchunk 0, $Q[1]$ drops the duplicate and keeps only 1, while 0 moves to $Q[2]$. As GPU 1 also forwards subchunk 3 to GPU 0, $Q[3]$ removes 3 and keeps only 2, while 3 moves to $Q[4]$. After SS (Fig. 7(b)), each subchunk is uniquely assigned: $Q[1]:\{1\}$, $Q[2]:\{0\}$, $Q[3]:\{2\}$, $Q[4]:\{3\}$. SS removes duplicates, picks lowest-delay paths, and finalizes unique assignments—avoiding redundancy while ensuring coverage.

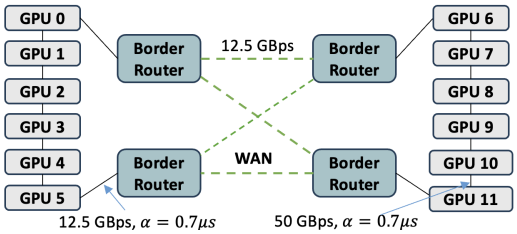

Figure 8: Minimal Connected Graph

5 Illustrative Numerical Results

In this section, we evaluate the performance of *SCALE-CCL* under various network configurations. By default, the simulated network adopts the following parameters: intra-DC GPU-to-GPU links operate at 50 GB/s bandwidth with 0.7 μs propagation delay, consistent with NVLink-level interconnects [15, 16]; GPU-to-border-router and WAN links have 12.5 GB/s bandwidth, representative of 100 Gbps Ethernet connections [9]. Intra-DC *connectivity* — defined as the ratio between the actual and fully connected number of intra-DC links — is varied to assess its impact. Unless otherwise specified, each GPU node generates the same amount of data, evenly partitioned into a fixed number of subchunks. The primary performance metric is the AllGather completion time, defined as the time required for all GPUs to receive all required subchunks.

We focus on three aspects: (1) the scalability of *SCALE-CCL* compared to existing approaches (TE-CCL), (2) the impact of subchunk granularity on overall performance, and (3) how varying WAN propagation delays affects the completion time across algorithms. All results are compared against three baselines: TE-CCL (an ILP-based global optimizer) [15], a shortest-path heuristic (SPH) that routes each subchunk along a minimum-latency path (Dijkstra over per-link delays) and serves packets in FIFO order without duplicate elimination or load-aware re-routing, and NCCL as an industry baseline. NCCL is evaluated with its default algorithm selection.

5.1 Scalability Analysis

Table 1 reports execution time of four schedulers (SCALE-CCL, TE-CCL, SPH, NCCL) under a 150 km WAN delay (0.75 ms) and intra-DC connectivity of 0.5[2]. Each GPU sends a 1–256 MB chunk, split into 1–8 subchunks, via `AllGather`. **SCALE-CCL** keeps execution time under 1 s in all settings, rising smoothly and sublinearly with subchunk count (e.g., 1 MB: 0.04–0.75 s). **TE-CCL** scales poorly: a

[2]Due to space limits, we only show results for connectivity = 0.5. Additional runs with 0.7/0.9 and WAN distances 10–200 km show consistent trends, with gaps always below 15%.

Table 1: Execution Time of Different Algorithm with 150km WAN Delay (Connectivity = 0.5)

Unit: s	SCALE-CCL				TE-CCL				Shortest Path				NCCL			
Num. Subchunks	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
1MB	0.04	0.08	0.22	0.75	4.61	35.24	1348.71	x	0.08	0.18	0.39	0.95	0.054	0.104	0.233	0.575
4MB	0.04	0.09	0.22	0.76	2.34	23.45	776.06	x	0.08	0.17	0.38	0.96	0.052	0.107	0.240	0.610
16MB	0.04	0.09	0.29	0.77	1.96	22.75	705.71	x	0.08	0.17	0.36	0.81	0.052	0.104	0.230	0.580
64MB	0.05	0.09	0.23	0.75	1.82	22.95	676.27	x	0.08	0.17	0.34	0.79	0.052	0.101	0.221	0.546
256MB	0.04	0.09	0.22	0.76	2.01	17.22	707.71	x	0.08	0.17	0.32	0.77	0.052	0.100	0.217	0.537

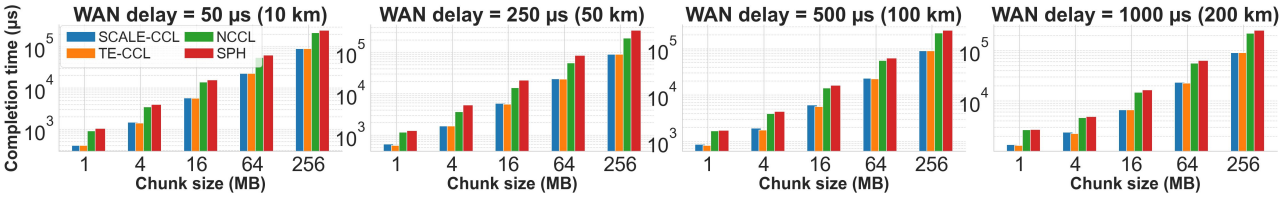

Figure 9: Completion Time vs. chunk size under *different WAN delay* with 0.5 connectivity and 1 Subchunk per GPU

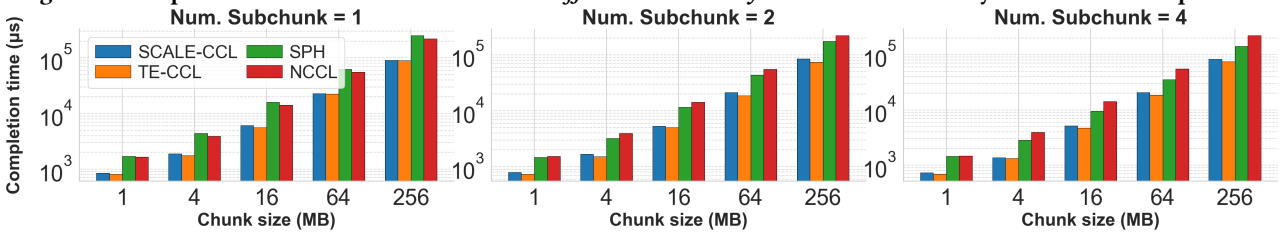

Figure 10: Completion Time vs. chunk size under *different number of subchunks* with 100 km WAN Delay and 0.5 connectivity

few seconds at 1 subchunk (2.01–4.61 s) but hundreds or thousands at 4, and timeouts at 8 ("x"). **SPH** is practical but 10–30% slower than SCALE-CCL in most cases (e.g., 1 MB, 8 subchunks: 0.95 s vs. 0.75 s), showing predictable growth. **NCCL** builds one ring and forwards all subchunks along it, avoiding per-subchunk path selection. Its time stays stable at 0.05–0.61 s and can be lower than SCALE-CCL at large subchunk counts (e.g., 256 MB, 8 subchunks: 0.537 s vs. 0.76 s), remaining competitive overall.

5.2 Impact of WAN Propagation Delay

Fig. 9 shows AllGather completion times under varying WAN propagation delays, from 50 μs (10 km) to 1000 μs (200 km), with intra-DC connectivity fixed at 0.5 and each GPU generating one subchunk. *SCALE-CCL* achieves near-optimal completion time with far lower scheduling overhead, where overhead is the offline synthesis time to generate a schedule policy; we quantify this reduction in the evaluation[2]. Across delays and chunk sizes, SCALE-CCL stays within 10% of TE-CCL (max 8.89%, avg 2.94%), confirming high-quality schedules without global ILP optimization. At 250 μs and 256 MB, SPH takes 340,062.6 μs and NCCL 220,251.4 μs, while SCALE-CCL and TE-CCL complete in 90,252.1 μs and 90,002.1 μs, making SCALE-CCL 3.8× faster than SPH, 2.4× faster than NCCL, and within 0.3% of TE-CCL. Completion time increases with propagation delay for all methods; SCALE-CCL remains close to TE-CCL, while SPH and NCCL grow more sharply.

5.3 Impact of Subchunk Count

Fig. 10 presents AllGather completion times under varying numbers of subchunks per GPU (1, 2, and 4) and chunk sizes from 1 MB to 256 MB. Intra-DC connectivity is fixed at 0.5 and WAN propagation delay at 500 μs. SCALE-CCL remains near-optimal across all 15 settings (3 subchunk counts × 5 chunk sizes), within 10% of TE-CCL in 12 cases and within 15% in all. For example, at 16 MB with 2 subchunks, SPH takes 11,566 μs and NCCL 14,251.4 μs, whereas SCALE-CCL and TE-CCL complete in 5,313.2 μs and 5,001.4 μs, respectively, making SCALE-CCL 2.18× faster than SPH, 2.68× faster than NCCL, and 6.2% slower than TE-CCL. NCCL is largely insensitive to subchunking at medium and large chunks because its ring AllGather serializes traffic over the same inter-DC bottleneck in each phase; increasing subchunk count changes granularity but not the bottleneck service time or queueing. Overall, higher subchunk counts reduce completion time for SCALE-CCL and TE-CCL with diminishing returns.

6 Conclusion and Future Work

In this paper, we introduced **SCALE-CCL**, a lightweight, load-aware collective communication library for cross-DC AI training over wide-area networks (WANs). Leveraging per-node link statistics, SCALE-CCL avoids the scalability and responsiveness limits of ILP-based schemes such as TE-CCL. Simulations show that SCALE-CCL achieves AllGather completion times within 15% of the near-optimal schedules produced by TE-CCL, while cutting scheduling time by up to *three orders of magnitude*. It consistently outperforms shortest-path heuristics (SPH) and generally surpasses NCCL, with only limited exceptions at high subchunk counts, across diverse chunk sizes, granularities, and WAN conditions.

While SCALE-CCL achieves promising results, several limitations and open challenges remain. First, our current evaluation assumes synchronized start times and homogeneous chunk sizes, which may not hold in asynchronous or heterogeneous training scenarios. Second, SCALE-CCL currently treats intra-DC topology as static and idealized. In real-world systems, localized failures or load imbalance could degrade performance [1]; incorporating this is an important additional direction for future work.

References

[1] Hitesh Allam. 2025. Reliability at the Edge: SRE for Distributed Cloud and IoT Platforms. *International Journal of Emerging Research in Engineering and Technology* 6, 2 (2025), 39–52.

[2] AMD ROCm Team. 2021. RCCL: Radeon Collective Communication Library. https://github.com/ROCm/rccl. Accessed: 2025-08-27.

[3] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 62–75.

[4] Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, et al. 2024. {MAST}: Global scheduling of {ML} training across {Geo-Distributed} datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 563–580.

[5] Lang Fan, Xiaoning Zhang, Yangming Zhao, Keshav Sood, and Shui Yu. 2023. Online training flow scheduling for geo-distributed machine learning jobs over heterogeneous and dynamic networks. *IEEE Transactions on Cognitive Communications and Networking* 10, 1 (2023), 277–291.

[6] Bita Fatemipour, Zhe Zhang, and Marc St-Hilaire. 2024. A Survey on Replica Transfer Optimization Schemes in Geographically Distributed Data Centers. *IEEE Transactions on Network and Service Management* (2024).

[7] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing* 22, 6 (1996), 789–828.

[8] William Gropp, Ewing Lusk, and Anthony Skjellum. 1999. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press.

[9] InfiniBand Trade Association. 2014. InfiniBand Trade Association Releases Updated Specification for Remote Direct Memory Access over Converged Ethernet (RoCE). https://www.infinibandta.org/infiniband-trade-association-releases-updated-specification-for-remote-direct-memory-access-over-converged-ethernet-roce/. Press Release, September 16, 2014.

[10] Intel Corporation. 2021. oneCCL: Collective Communications Library. https://uxlfoundation.github.io/oneCCL/index.html. Accessed: 2025-08-27.

[11] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 3–14.

[12] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. 2008. Floodless in seattle: a scalable ethernet architecture for large enterprises. *ACM SIGCOMM Computer Communication Review* 38, 4 (2008), 3–14.

[13] Meng Lian, Yongli Zhao, Yike Jiang, Tingting Bao, Yuan Cao, and Jie Zhang. 2025. Software-defined cloud–optical networks for long-haul geographically distributed machine learning. *Journal of Optical Communications and Networking* 17, 5 (2025), 363–377.

[14] Meng Lian, Yongli Zhao, Xin Li, Wenhong Liu, Yajie Li, Massimo Tornatore, and Jie Zhang. 2025. Resource Allocation in Flexible-Bandwidth Fine-Grained Optical Transport Networks for Geo-Distributed Machine Learning. *IEEE Internet of Things Journal* (2025).

[15] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. 2024. Rethinking machine learning collective communication as a multi-commodity flow problem. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 16–37.

[16] J NVIDIA. 2022. NVIDIA NVLink high-speed GPU interconnect. (2022).

[17] Chetan Phalak, Dheeraj Chahal, Manju Ramesh, and Rekha Singhal. 2024. Towards Geo-Distributed Training of ML Models in a Multi-Cloud Environment. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*. 211–217.

[18] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[19] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding collective algorithm synthesis using communication sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 593–612.

[20] Won Wook Song, Myeongjae Jeon, and Byung-Gon Chun. 2022. SWAN: WAN-aware stream processing on geographically-distributed clusters. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*. 78–84.

[21] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. 2017. Distributed mean estimation with limited communication. In *International conference on machine learning*. PMLR, 3329–3337.

[22] Yongji Wu, Yechen Xu, Jingrong Chen, Zhaodong Wang, Ying Zhang, Matthew Lentz, and Danyang Zhuo. 2024. MCCS: A Service-based Approach to Collective Communication for Multi-Tenant Cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 679–690.