DSC-204A: Scalable Data Systems, Fall 2025

7: Basics of Cloud Computing - II

Lecturer: Hao Zhang *Scribe(s): Teresa Lee, Mia Lai, Ksheer Agrawal, Harshit Timmanagoudar*

1 Collective Communication

1.1 Motivation

Why Collective Communication?

All workers often need to communicate in the same pattern (e.g., averaging gradients). Instead of each worker using ad-hoc point-to-point communication, collective communication allows structured, efficient data exchange.

High-impact uses and implementations:

- Major machine-learning workloads (e.g., distributed training) rely on collectives. Supercomputers are a primary environment for collective communication. They provide the high-performance interconnects, NIC features, and MPI implementations(OpenMPI) that enable highly-optimized collective primitives.
- Companies optimize these operations aggressively. For example: NVIDIA uses NCCL (NVIDIA Collective Communications Library).Collective communication is an active HPC research topic because these optimizations affect large-scale performance.

Why is collective better than P2P?

Although collective communication involves many participants, it is optimized to reduce total data movement. For example, naive Allreduce might require $6n$ of raw send/receive volume, but optimized algorithms eliminate redundant transfers and exploit parallelism to reduce the effective volume (roughly to the order of $3n$), while also lowering wall-clock time with concurrent transfers.

Example: The latest **DeepSeek-V3 MoE** model uses All-to-All to route tokens among experts. All-to-All is bandwidth-heavy, so careful batching and topology-awareness are required to keep latency acceptable.

1.2 Fault Tolerance

Collectives are **not fault-tolerant**. If one worker fails, the entire operation must restart. By contrast, systems like **Ray** include checkpointing and recovery, allowing partial recomputation. This remains a challenge in large-scale distributed ML systems.

1.3 (α–β) Model

The cost of sending a message of size n bytes is commonly modeled as

$$T(n) = \alpha + n\beta,$$

where α (startup latency) is the per-message fixed cost and β is the per-byte transmission cost (inverse bandwidth). Intuitively:

- For very small messages α dominates (latency-limited regime).
- For large messages the $n\beta$ term dominates (bandwidth-limited regime).

1.4 Minimal-Spanning Tree (MST) Algorithm

Used for small-message collectives. Emphasizes low latency over bandwidth.

Each communication step halves the remaining nodes needing data transfer, leading to $\lceil \log p \rceil$ rounds. MST prioritizes minimizing latency but can under-utilize bandwidth since not all links are active simultaneously.

Why can't we just transfer all the messages together?

Answer: Because simultaneous transfers would saturate shared links and cause congestion. The MST structure schedules transfers to avoid overloading any single link while keeping latency low.

2 Broadcast

After communication, all workers hold the same message from the root.

$$T_{\text{bcast}} = \log(p)(\alpha + n\beta)$$

Example: In MST broadcast, the root sends to half the workers, who in turn relay to others. Each communication step incurs startup latency α and transmission cost proportional to $n\beta$.

Proof Intuition

Each round doubles the number of informed workers. After $\log p$ rounds, all are updated. Hence total cost = $\log(p)(\alpha + n\beta)$.

3 Reduce (to-one)

Aggregates data from multiple workers to one root (e.g., summing gradients).

$$T_{\text{reduce}} = \log(p)(\alpha + n\beta + n\gamma)$$

where γ is the per-byte computation cost for aggregation.

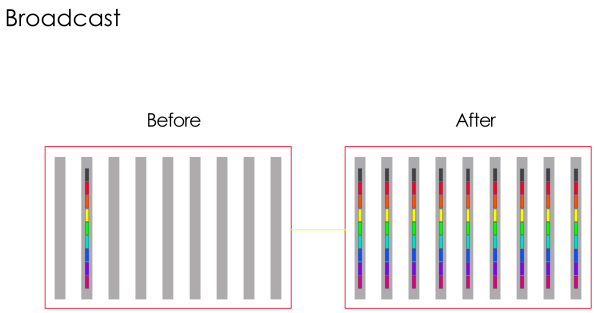

Figure 1: Broadcast using MST topology.

Concept: The tree is reversed — workers send their results up the hierarchy until the root has the combined result.

Scribe Note

Divide workers recursively into halves, sum partial results at each stage, continue until one node remains. Total cost = $\log(p)(\alpha + n\beta) + n\gamma$

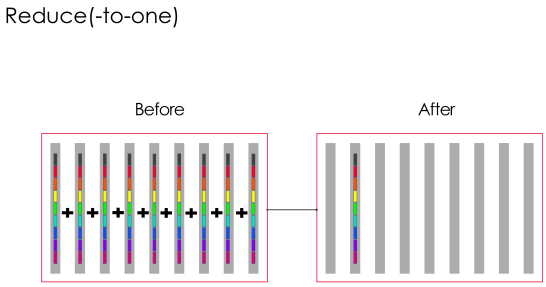

Figure 2: Reduce(-to-one) process using recursive halving.

4 Scatter

The scatter collective communication operation takes a contiguous set of data from one root process and distributes a different, equal chunk of that data to every process in the group.

Process steps:

- **Initial partition and sending**:
  The root divides its remaining data buffer into two equal halves. It keeps the first half for itself and

its portion of the group, and sends the second half to a designated partner process that will act as the new sub-root for the other half of the processes.

- **Recursive Halving and Forwarding**:
  Every process that just received a non-final chunk of data immediately becomes a temporary sub-root. It divides the data it received in half, keeps the first half for its subgroup, and sends the second half to its new partner. Multiple such communications occur in parallel across the network.
- **Termination**:
  The process continues until every participating process has received its final, smallest, and correct chunk of the original data.

$$T_{scatter} = \sum_{k=1}^{log(p)} \left(\alpha + \frac{n}{2^k}\beta\right) = \log(p)\alpha + \frac{p-1}{p}n\beta$$

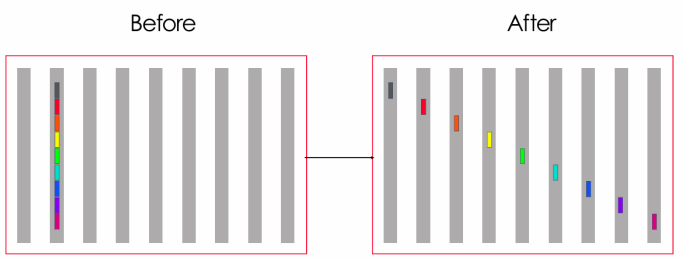

Figure 3: Scatter communication.

5 Gather

The gather collective communication operation is essentially the reverse of the scatter operation.

Process steps:

- **Initial partition and receiving**:
  The non-receiving processes send their data chunk to a designated partner process (a sub-root), which then concatenates the incoming data with its own to form a larger, contiguous block.
- **Recursive Merging and Forwarding**:
  The new sub-roots that have accumulated data now act as senders to the next level up the tree. They combine all the data they currently hold (which is already a consolidated buffer) and send this doubled-in-size buffer to their new partner, closer to the final root.
- **Termination**:
  The operation culminates when the root process receives the last large, accumulated buffer from its final partner in the tree.

$$T_{gather} = \log(p)\alpha + \frac{p-1}{p}n\beta$$

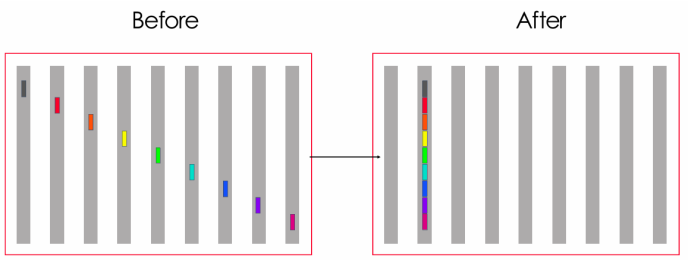

Figure 4: Gather communication.

6 All Gather

What is All Gather?

All Gather is the combination of **Gather** and **Broadcast** communication styles. Basically, all nodes have a fraction of the message and we need all the nodes to have the whole message. This is essential for synchronizing states in parallel training.

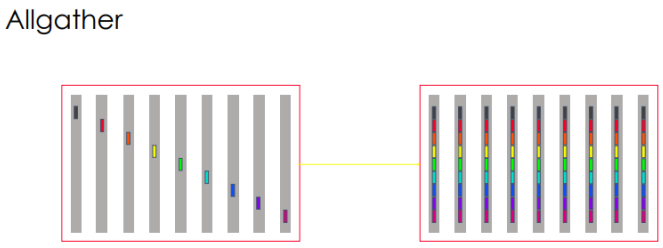

Figure 5: Overview of the All Gather communication

Breakdown of All Gather steps into **Gather** and **Broadcast**:

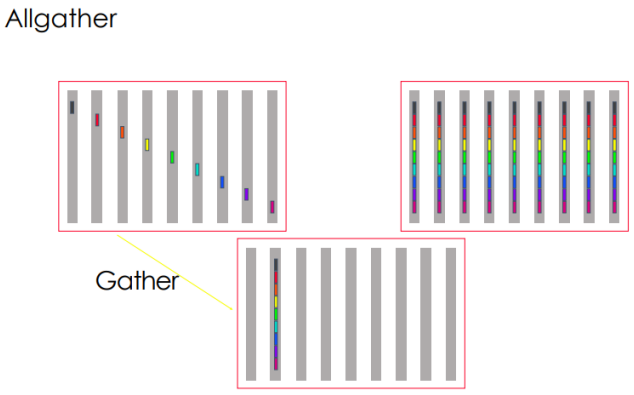

Figure 6: Gather in All Gather

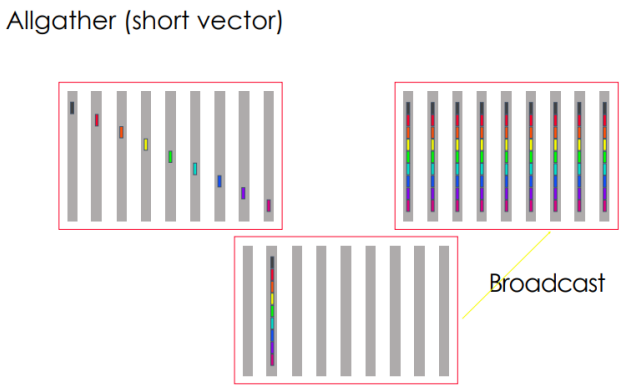

Figure 7: Broadcast in All Gather

To find the latency of All Gather, sum the latency of Gather and Broadcast:

$$T_{gather} = \log(p)\alpha + \frac{p-1}{p}n\beta$$

$$T_{broadcast} = \log(p)(\alpha + n\beta)$$

$$T_{allGather} = T_{gather} + T_{broadcast} = 2\log(p)\alpha + (\frac{p-1}{p} + \log(p))n\beta$$

7 Reduce-scatter

Reduce scatter starts with every node having the full message, and ends with each node having just a fraction of the full message. Reduce-scatter is a combination of **Reduce** and **Scatter** communication styles. it performs partial reduction followed by scattering results back.

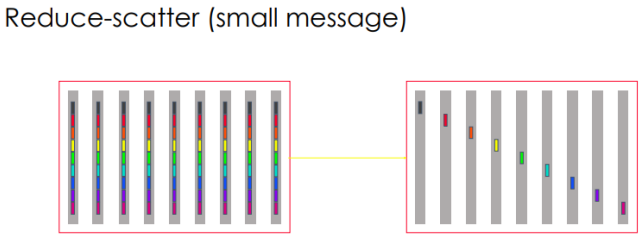

Figure 8: Overview of the Reduce-scatter communication

Breakdown of Reduce-scatter steps into **Reduce** and **Scatter**:

To find the latency of Reduce-scatter, sum the latency of Reduce and Scatter:

$$T_{reduce} = \log(p)(\alpha + n\beta + n\gamma)$$

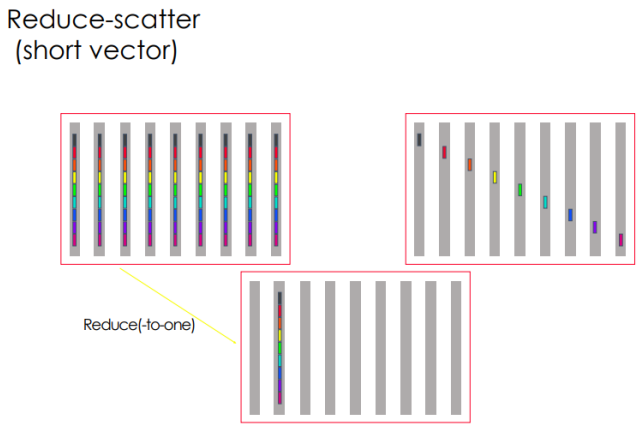

Figure 9: Reduce in reduce scatter

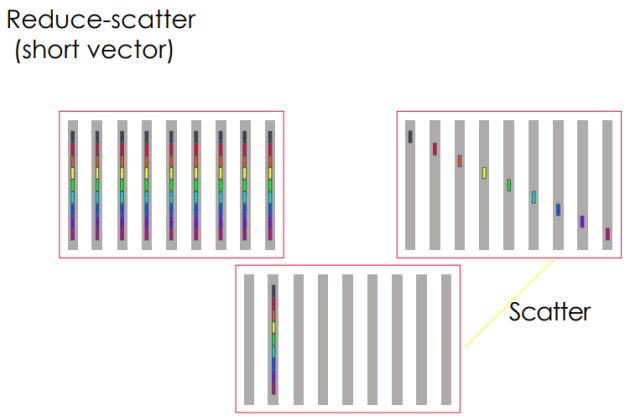

Figure 10: Scatter in reduce scatter

$$T_{scatter} = \log(p)\alpha + \frac{p-1}{p}n\beta$$

$$T_{reduceScatter} = T_{reduce} + T_{scatter} = 2\log(p)\alpha + (\frac{p-1}{p} + \log(p))n\beta + \log(p)n\gamma$$

8 Allreduce

Allreduce combines the results from all the ranks and sends a copy of the aggregated result back to all the ranks.

- Step 1 (Reduce-to-one): Using a binary tree structure, workers aggregate data upward to a root node.
- Step 2 (Broadcast): The root node distributes the aggregated result back to all workers using the same tree structure.

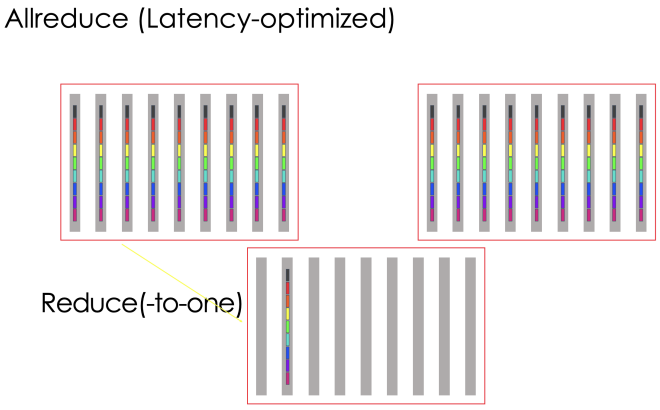

Figure 11: Allreduce. Step 1(Reduce-to-one).

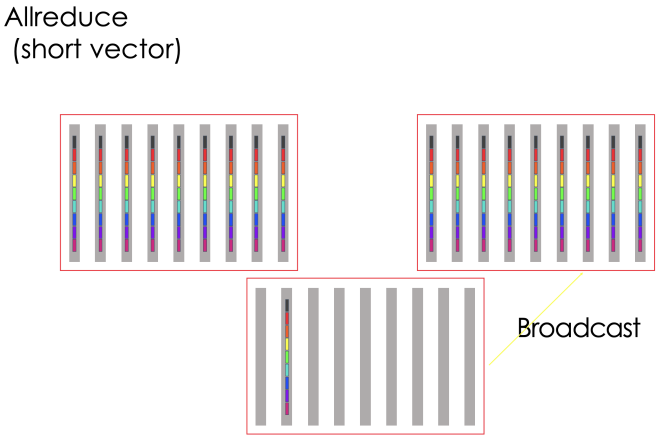

Figure 12: Allreduce. Step 2(Broadcast).

8.1 Importance in Machine Learning

Allreduce is the **most critical** collective communication operation in distributed machine learning training:

1. Each GPU processes a mini-batch and computes local gradients: $\nabla L(\theta, D_i)$
2. All GPUs need the average gradient: $\frac{1}{p}\sum_{i=0}^{p-1}\nabla L(\theta, D_i)$
3. Allreduce aggregates all gradients (sum operation)
4. All workers update synchronously: $\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{p}\text{Allreduce}(\nabla L(\theta, D_i))$

Without Allreduce, each worker would need to send gradients to every other worker ($O(p^2)$ messages), making the process highly inefficient.

8.2 Cost Analysis

The total cost of Allreduce combines the costs of both phases:

Formula Breakdown:

- $2\log(p)\alpha$: Latency cost for $2\log(p)$ communication rounds (log(p) rounds each for Reduce-to-one and Broadcast)
- $\log(p)n \cdot 2\beta$: Bandwidth cost for data transmission during both phases
- $\log(p)n \cdot \gamma$: Computation cost for performing the reduction operations at each level of the tree

Total Cost:

$$T_{allreduce} = 2\log(p)\alpha + \log(p)n(2\beta + \gamma)$$

This formula represents the **latency-optimized** (Minimum Spanning Tree) approach, which minimizes the number of communication rounds, making it suitable for small messages where latency dominates over bandwidth.

9 All-to-All

All-to-All is a collective communication operation where each rank sends **unique, personalized** data to every other rank simultaneously. Unlike Allreduce or Allgather where all ranks receive the same result, in All-to-All, each rank receives different data from each sender.

9.1 Operation Description

- **Before**: Each worker has p distinct data chunks, where chunk j is destined for worker j
- **After**: Each worker has collected its designated chunk from all p workers

Concrete Example with 4 workers:

- Worker 0 sends: chunk A to worker 1, chunk B to worker 2, chunk C to worker 3
- Worker 1 sends: chunk D to worker 0, chunk E to worker 2, chunk F to worker 3
- Worker 2 sends: chunk G to worker 0, chunk H to worker 1, chunk I to worker 3
- Worker 3 sends: chunk J to worker 0, chunk K to worker 1, chunk L to worker 2

After All-to-All, Worker 0 has received chunks {D, G, J}, Worker 1 has {A, H, K}, and so on.

9.2 Characteristics

All-to-All is the **most communication-intensive** collective operation:

- Requires $p(p-1)$ simultaneous point-to-point communications

- Every network link must be utilized, making it bandwidth-limited
- Commonly used in distributed matrix transpose and FFT algorithms
- Highly sensitive to network congestion and topology

Each node sends unique data to every other node. Highly communication-intensive.

$$T_{all2all} = (p-1)(\alpha + n\beta)$$

Optimizing this typically requires advanced scheduling or pipelining to avoid congestion.

10 Recap: MST Algorithm Costs

The following table summarizes the communication costs for all collective operations using the Minimum Spanning Tree (latency-optimized) approach:

Operation	**Cost**
Broadcast	$\log(p)\alpha + n\beta$
Reduce(-to-one)	$\log(p)(\alpha + n\beta + n\beta)$
Scatter	$\log(p)\alpha + \frac{p-1}{p}n\beta$
Gather	$\log(p)\alpha + \frac{p-1}{p}n\beta$
Allgather	$2\log(p)\alpha + \log(p)n\beta + \frac{p-1}{p}n\beta$
Reduce-scatter	$2\log(p)\alpha + \log(p)n(\beta+\gamma) + \frac{p-1}{p}n\beta + \log(p)n\beta$
Allreduce	$2\log(p)\alpha + \log(p)n(2\beta+\gamma)$

Table 1: Summary of MST collective communication costs.

All operations exhibit $\log(p)$ latency terms, which reflects the binary tree structure requiring $\log(p)$ levels of communication. Operations built from simpler primitives, such as Allreduce being composed of Reduce-to-one and Broadcast, have costs that are simply additive. The term $\frac{p-1}{p}n\beta$ appears in Scatter and Gather operations due to the progressive halving of data at each tree level. The γ parameter represents the computation cost incurred during reduction operations where data must be combined.

11 Summary of MST Algorithms

The MST Algorithms are designed mainly for small messages where performance is determined mostly by the latency. It focuses on reducing the communication rounds by following a binary tree structure for orgainizing the workers/ranks. This approach is optimal when message size $n \to 0$, placing the system in the latency-dominated regime of the $\alpha\beta$ model.

The problem that arises with this is that the bandwidth utilization may be overlooked in certain communications such as the large messages. To handle large message scenarios effectively, we need a different algorithmic approach:

12 Large Message Algorithms

12.1 Communication Model Revisited

In the $\alpha\beta$ model where Cost $= \alpha + n\beta$ and $\beta = \frac{1}{B}$ (the inverse of bandwidth), the behavior changes dramatically as message size grows. For large messages where $n \to \infty$, the bandwidth term $n\beta$ dominates the overall cost. This shifts our optimization goal from minimizing communication rounds to maximizing bandwidth utilization by keeping all network links active simultaneously.

12.2 General Principles: Ring Algorithm

The ring algorithm addresses the bandwidth utilization problem by fundamentally changing the communication topology. Instead of using a tree structure, nodes are arranged in a logical ring where each node communicates with its immediate neighbors. This allows all network links to operate simultaneously, passing data chunks around the ring in a coordinated manner.

The logical ring can be efficiently embedded in a physical linear array using worm-hole routing techniques. Importantly, the "wrap-around" connection that links the last node back to the first node doesn't create communication conflicts, allowing all p links to operate concurrently and achieve full bandwidth utilization.

12.3 Trade-offs: MST vs Ring

The choice between MST and Ring algorithms depends on message characteristics. MST algorithms excel with small messages due to their minimal $\log(p)$ communication rounds, making them ideal for the α-dominated regime. Ring algorithms, while requiring more rounds, achieve full bandwidth utilization, making them superior for large messages in the $n\beta$-dominated regime.

Modern high-performance communication libraries such as NCCL, MCCL, and OneCCL implement both algorithmic families and automatically select the appropriate one based on message size and network topology. The selection threshold typically falls between 1KB and 1MB, depending on specific system characteristics and the relative costs of latency versus bandwidth in the deployment environment.