DSC-204A: Scalable Data Systems, Fall 2025

8: Collective Communication 2, Data Warehouse

Lecturer: Hao Zhang *Scribe: Avinash Gondela, Aditi Pagey, Sibo Zhu, Isheta Bansal*

1 Continuation about collective communication

The communication model under discussion is the **Alpha-Beta Model**. Its formula is given by:

$$Communication\ Model = \alpha + n \times \beta$$

where

$$\beta = \frac{1}{B}$$

The second term, $n \times \beta$, dominates; hence, we aim to minimize this term to utilize the bandwidth as efficiently as possible.

2 Benefits of Ring Algorithm

- It utilizes the bandwidth properly, and at any time any connection between any two nodes works
- It can be implemented for any random number of nodes

3 Bandwidth Utilization and MST Preference

Important: In machine learning, the **Minimum Spanning Tree (MST)** is often preferred because it fully utilizes the available bandwidth.

4 Primitives in Ring Algorithm

4.1 All Gather

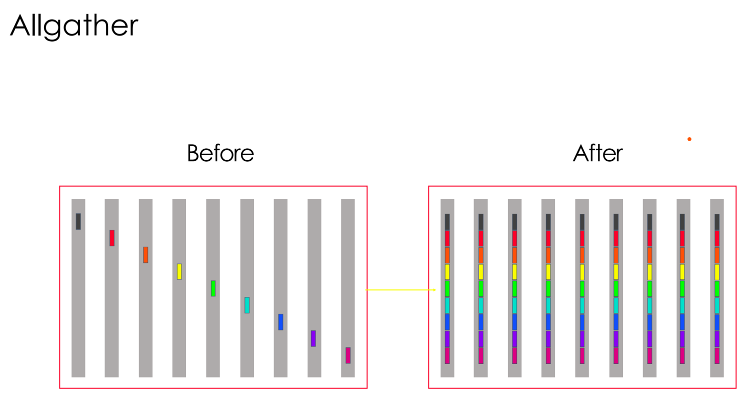


Figure 1: All Gather in Ring Algorithm

Cost of All Gather is given by:

$$(p-1)\left(\alpha + \frac{n}{p} \times \beta\right)$$

4.2 Reduce Scatter

This is not a reduce + scatter, instead it becomes a primitive.

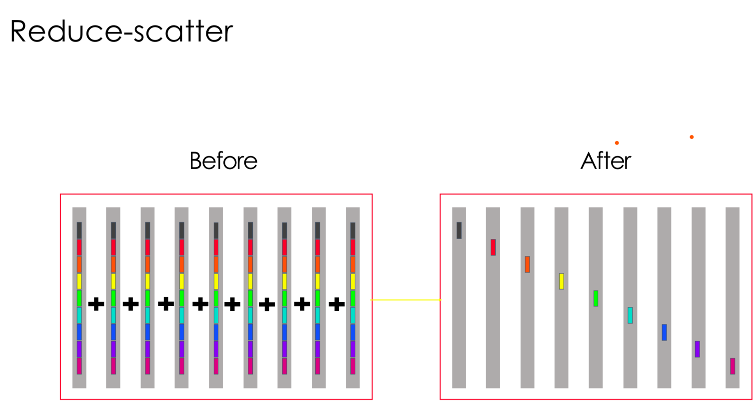


Figure 2: Reduce Scatter in Ring Algorithm

Cost of Reduce Scatter is given by:

$$(p-1)\left(\alpha+\frac{n}{p}\cdot\beta+\frac{n}{p}\cdot\gamma\right)$$

4.3 Reduce Scatter vs All Gather

- **Important:** The **Reduce Scatter** operation (reduce followed by scatter) incurs more latency than **All Gather** due to an additional step involving the introduction of a γ term.
- For any primitive that requires performing a reduction, there exists an additional γ term. For long messages, this term becomes expensive because it represents an extra computation requiring significant FLOPs. Thus, it demands compute power from CPUs or GPUs.
- In the context of machine learning, this implies allocating **Streaming Multiprocessors (SMs)** from GPUs, where an SM represents a computing unit. For example, NVIDIA's H100 GPU contains nearly 130 SMs. This introduces a **compute bottleneck** in addition to the bandwidth bottleneck.

To alleviate this, we aim to minimize the γ term. Modern GPU architectures include dedicated hardware units specifically designed to process γ. Similarly, network chips now include specialized components for γ-reduction operations. This represents a **co-design approach** between networking and machine learning systems. Check the concept of NVSHARP!

Question: Can the Ring algorithm perform better? In other words, how can we prove that scatter or gather using MST is most optimal in terms of bandwidth?

Answer: MST is better because we always prefer direct communication to the destination cell instead of sequential (hop-by-hop) transmission which is useful for the case of scatter and gather.

5 Building Blocks for Broadcast (Large Messages)

- In MST, **broadcast** is a primitive operation. In the ring topology, it is implemented as **scatter + all-gather**, so the latency is the sum of both.
- In MST, **reduce** is a primitive operation. In the ring topology, it is implemented as **reduce-scatter + gather**.
- All reduce can be termed as the sum of Reduce Scatter and All Gather

On a high level, we prefer **reduce-scatter + all-gather** for All Reduce because it offers higher bandwidth utilization and lower memory consumption.

5.1 Summary of Collective Communication Primitives

1. Reduce = Reduce-scatter + Gather
2. All-reduce = Reduce-scatter + All-gather
3. Broadcast = Scatter + All-gather

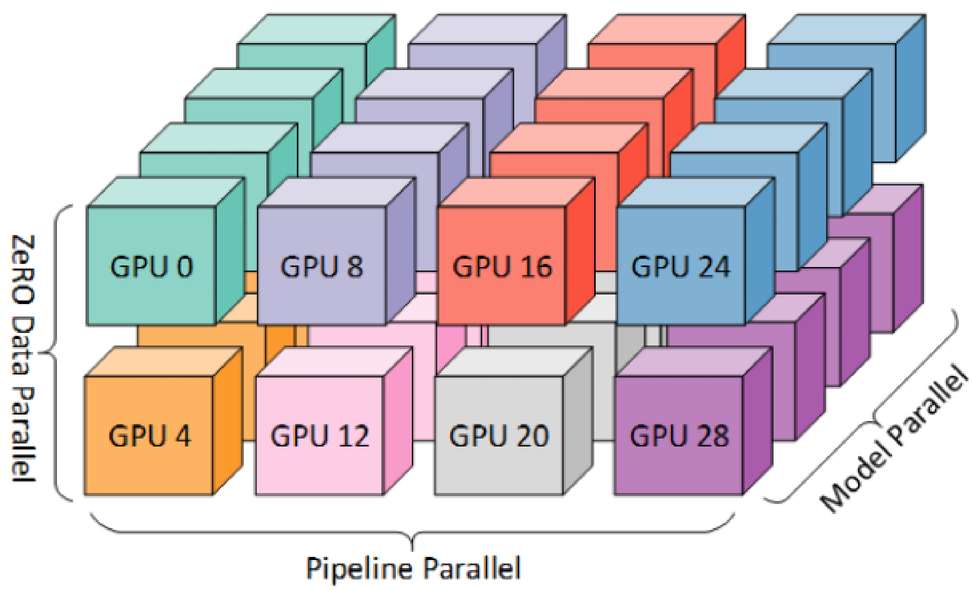
6 Real Cluster Setup for Training ChatGPT

6.1 Hardware (H200 GPUs)

Each compute node is equipped with up to eight NVIDIA H200 GPUs. To scale up training capacity, additional nodes are added to form a larger distributed cluster.

6.2 Interconnect and Communication

Within a single node, the eight GPUs are connected through a high-speed interconnect (such as NVLink or NVSwitch), which enables fast data exchange and efficient parallel computation. Across the cluster, multiple nodes are interconnected—typically through a high-bandwidth network topology. Communication within the same interconnect group is very fast, whereas communication between different groups occurs over links with comparatively lower bandwidth.



6.3 GPU vs TPU Interconnects

• **GPU Clusters:** Typically use a 2D mesh topology, connecting GPUs in rows and columns for efficient communication.

• **TPU Clusters:** Use a 3D mesh, adding an extra dimension of connectivity for better scalability and parallelism.

• Hierarchy of Meshes: A 1D mesh can be extended into a 2D mesh, and a 2D mesh can be expanded into a 3D mesh, allowing larger clusters to communicate efficiently.

7 Collective Communication

7.1 Pros:

• Structured and well-defined communication primitives – makes the communication pattern predictable and organized.

• Well optimized – designed for efficiency, ensuring data moves quickly between nodes.

• Mathematically analyzable – performance can be understood and predicted using formal analysis.

7.2 Cons:

• Lack of fault tolerance – if a single node fails, the entire collective operation fails, which can disrupt computations.

• Requires homogeneity – performs best when all nodes are identical and all bandwidths are equal, which is rarely achievable in real-world systems.

8 Operations of Database

8.1 CRUD:

• **Create:** Add new data to the database.

• **Read:** Retrieve existing data.

• **Update:** Modify existing data.

• **Delete:** Remove data from the database.

8.2 Online Transaction Processing (OLTP)

• Reads only a small number of records per query, supporting fast transactions.

• Provides low-latency, random-access writes for real-time updates.

• Used by end users or customers through web applications.

• Represents the current state of the data.

• Typically handles data sizes from gigabytes to terabytes.

8.3 Online Analytical Processing (OLAP)

• Reads and aggregates large volumes of records to support analysis.

• Handles bulk imports or streaming writes via ETL processes.

• Used primarily by analysts for decision-making and insights.

• Stores historical events and data for long-term analysis.

• Typically deals with data sizes from terabytes to petabytes.

8.4 OLTP Data Considerations

• OLTP data must be highly available and carefully protected to ensure transactional integrity.

• An additional layer is needed on top of the database to prevent accidental corruption when multiple users are accessing and updating the data simultaneously.

• Any reads for analysis should be performed on a read-only copy to avoid affecting the live transactional data.

9 Data Warehousing

- **Data Warehouse** solves the problem of running analytics on live transactional data.
- It acts as a separate layer built on top of OLTP systems.
- It stores a read-only copy of data for analytic queries.
- It prevents heavy analytic queries from slowing down OLTP systems.
- It ensures data consistency for multiple analysts working at once.
- Data warehouses are common in large enterprises, small companies might use simpler solutions, such as **Levels.fyi** which used Google Sheets as a backend to scale to millions of users.
- It maintains low latency and supports ad-hoc analytic queries.

9.1 Extract-Transform-Load (ETL)

This is the common process used to populate a data warehouse:

- **Extract:** Get data from the various OLTP databases (e.g., periodic data dumps or continuous streaming).
- **Transform:** Convert the data into an analysis-friendly schema. This step includes data cleaning and restructuring.
- **Load:** Store the transformed data into the data warehouse.

9.2 Why use a Data Warehouse?

- **Separation of concerns:** Keeps OLTP and OLAP workloads isolated. It allows OLTP systems to focus on low-latency transactions while OLAP systems handle complex analytical queries. This separation ensures performance, reliability, and manageable latency for both.
- **Expertise and Management:** The systems require different management and expertise. It is easier to maintain and scale independently.

- **Optimized for Analytics:** Classic indexes used in OLTP (like SSTables, B-trees) are good for reading/writing single records but are not efficient for answering large-scale analytic queries. Data warehouses use different storage strategies (e.g., column-oriented storage).

9.3 Interaction with OLAP and OLTP

- Both OLAP and OLTP systems can often be interacted with using a **SQL query interface**.
- For OLAP, there is a growing trend of codeless user interfaces, including **Text2SQL** capabilities.

10 Distributed Computing and Big Data

10.1 Core Concepts

This area of study covers several key topics:

- Parallelism Basics
- Data Replication and Partitioning
- Batched Processing
- Streaming Processing

10.2 Parallel Data Processing

- **Central Issue:** The workload takes too long for a single processor to handle.
- **Basic Idea:** Split the workload across multiple processors and machines. This is achieved through a **Divide and Conquer** strategy.
  - **Divide and Conquer Strategy**: This strategy divides one problem into small sub-problems. This division keeps taking place until one processor can finish the work in an acceptable amount of time. This gives us a partial result. These partial results are merged back to form the final result for the main problem.

10.3 Data Processing Abstraction

- We need a way to represent various processing functions (e.g., sum, mean, PageRank, supervised learning, model inference).
- **Question:** How can we represent these arbitrarily complex processing functions?

10.4 Dataflow Graphs

- A **Dataflow Graph** is a common abstraction in parallel data processing.
- It is a directed graph representing a program:

  - **Vertices (Nodes):** Abstract operations (computational primitives).
  - **Edges:** Represent the direction of data flow, showing data dependency.

- **Examples:**
  - **Relational Dataflows:** Used in RDBMS, Pandas. This is also known as a **Logical Query Plan** in database systems.
  - **Matrix/Tensor Dataflows:** Used in NumPy, PyTorch, TensorFlow. This is also known as a **Neural Network Computational Graph** in ML systems.

10.5 Parallelism Paradigms

Key parallelism paradigms in data systems (assuming coordination):

- **Task Parallelism:** Different functions are executed on the same or different data.
- **Data Parallelism:** The same function is executed on different partitions of the data.
- **Hybrid Parallelism:** A combination of both task and data parallelism.