DSC204A: Scaleable Data Systems Winter 2025

Lecture 20: Collective Communication

Instructor: Umesh Bellur *Scribe: Chirag Agarwal, Tristan Cooper, Varadraj Bartakke*

1 Recap

Question 1 - **Protocols specify the implementation**

Answer: False
Explanation: Protocols define the rules and expected behaviors for communication between systems, such as message formats and sequences. They do not dictate how these rules are implemented in code or hardware. Implementation details are left up to the developers.

Question 2 - **Congestion control takes care of the sender not overflowing the receiver**

Answer: False
Explanation: Congestion control is concerned with preventing excessive traffic from overwhelming the core of the network, which may have multiple senders and receivers. It aims to avoid issues like packet drops due to overloaded routers. On the other hand, flow control manages the rate at which a sender transmits data to ensure the receiver is not overwhelmed.

Question 3 - **A random access protocol is efficient at low utilization**

Answer: True
Explanation: Random access protocols allow hosts to transmit data whenever they have something to send, without prior coordination. If two transmissions collide, the protocol includes a method for detecting or recovering from the collision. At low utilization, when few hosts are trying to transmit simultaneously, collisions are rare, making the protocol efficient and responsive.

Question 4 - **At the data link layer, hosts are identified by IP addresses**

Answer: False
Explanation: IP addresses are part of the network layer and are used for logical addressing and routing across networks. The data link layer, on the other hand, deals with physical or MAC (Media Access Control) addresses, which are used for communication within the same local network segment.

Question 5 - **The physical layer is concerned with sending and receiving bits**

Answer: True
Explanation: The physical layer is the lowest layer in the network stack and is responsible for the actual transmission and reception of raw bits over a physical medium. It deals with hardware components, signal encoding, and transmission media.

Question 6 - **Layering improves application performance**

Answer: False
Explanation: Layering introduces overhead due to the addition of headers, encapsulation, and processing at each layer.

However, layering is used in networking because it promotes modularity: if a lower layer changes (e.g., switching from Ethernet to Wi-Fi), only the adjacent upper layer may need to adapt, while the rest

of the stack remains unaffected. This separation of concerns simplifies development, maintenance, and interoperability.

Question 7 - **Routers forward a packet based on its destination address**

Answer: True
Explanation: Routers use the destination IP address in the packet header to determine the best path to forward the packet through the network toward its destination. They maintain routing tables that map destination addresses to outgoing interfaces.

Question 8 - **"Best Effort" packet delivery ensures that packets are delivered in order**

Answer: False
Explanation: Best Effort delivery means the network tries to deliver packets but provides no guarantees on delivery, order, or duplication. Packets may arrive out of order, be lost, or duplicated, as no mechanisms ensure reliable or ordered delivery.

Question 9 - **Port numbers belong to network layer**

Answer: False
Explanation: Port numbers are part of the application layer and are used to identify specific applications or services on a host. The network layer deals with logical addressing such as IP addresses, which route packets between hosts.

2 Point-to-point Communication

Point-to-point communication is a direct data transmission between two network nodes, typically a sender and a receiver.

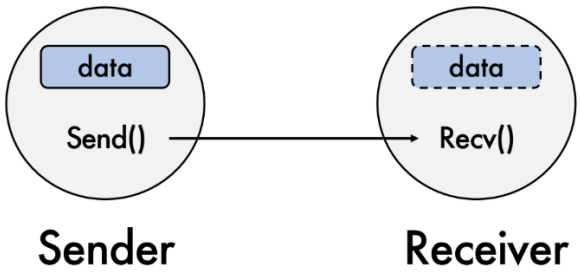

Figure 1: Point-to-point Communication between two nodes

1. A TCP connection is established between the sender and receiver.
2. The application on the sender side sends data to be transmitted.
3. The data passes down through the five layers of the network stack (Application, Transport, Network, Data Link, Physical), travels across the network, and arrives at the receiver where it moves up through the layers for processing.

2.1 P2P Communication in Ray

Ray enables efficient distributed computing by passing references to objects in a shared object store, allowing tasks to communicate in a peer-to-peer manner without copying large data.

```
import ray

# Initialize Ray
ray.init(ignore_reinit_error=True)

# Define a simple remote function
@ray.remote
def process_data(data_ref):
    # Access the object referenced by data_ref
    data = ray.get(data_ref)
    return [x * 2 for x in data]

# Put an object into Ray's object store
data = [1, 2, 3, 4, 5]
data_ref = ray.put(data)  # Get object reference (ObjectID)

# Pass the object reference to another process/task
result_ref = process_data.remote(data_ref)  # Task with object reference

# Get the result from the remote task
result = ray.get(result_ref)

# Print the result
print("Processed data:", result)
```

- `ray.put(data)` stores the data object in Ray's distributed object store and returns an object reference (`data_ref`).
- `process_data.remote(data_ref)` launches a remote task, passing the object reference instead of the actual data.
- `ray.get(result_ref)` retrieves the result of the remote computation.

3 Stochastic Gradient update

Stochastic Gradient descent is one example of an application of collective computing.

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

objective data

Figure 2: SGD Update Rule

Calculating the SGD update can be parallelized by splitting up the data between different machines. Each machine is given a part of the data to calculate its portion of the gradient and sends it to the other machines. Once all machines are done with this part, they can then calculate the global gradient and use it to update the parameter values.

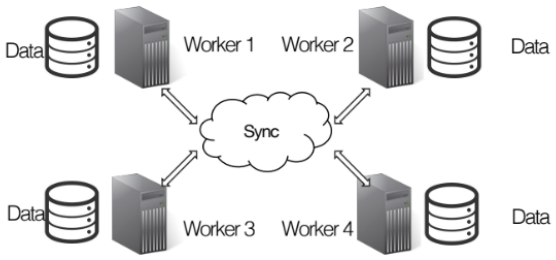

Figure 3: SGD Update Process Using Multiple Machines

This has significant communication and synchronization overheads. Communication overhead is the time and resources spent sending messages between machines. Each machine needs to send a message to every other one, so as the number of machines grows, so does the work each machine does. This causes issues with scalability.

Synchronization overhead is caused by different workers being slower than others, or some being scheduled later than others. Since the machines need to wait for all of them to complete calculating their gradient in order to calculate the global gradient, some of the machines may be idle for some time.

4 Collective Communication Primitives

4.1 Reduce

We want a single primitive which lets a machine receive the gradients from every other machine. We call this reduce.

This opens up a different approach, where one machine is given all of the partial gradients, adds it up, and gives the global gradient to all other machines. This only requires $O(N)$ communication overhead rather than $O(N^2)$, but also means there is a single point of failure, which is not desirable.

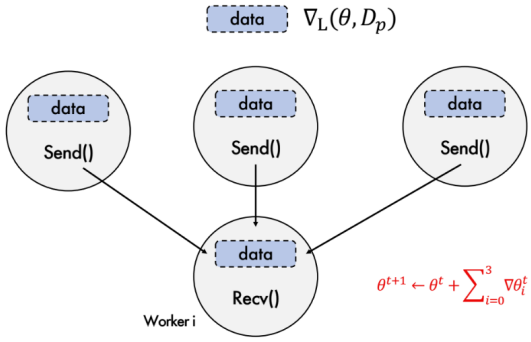

Figure 4: The Reduce Primitive

4.2 All-Reduce

In SGD, it's okay if some of the machines fail, since losing some of the partial gradients will only cause a slight loss in accuracy for calculating the global gradient, so we'd like to use a distributed approach rather than centralized. To do this, we use a new primitive all-reduce.

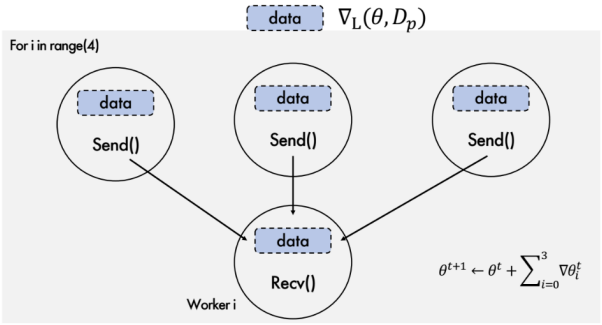

Figure 5: All-Reduce

All-reduce gathers all of the data from the other machines and applies some function to the data, all in one step. The function can be anything, such as a weighted average or a sum.

Overall, using collective communication libraries is convenient because using primitives can simplify coding, and libraries can be optimized very thoroughly.

5 Model of Parallel Computation

In this model, the network is assumed to be *fully connected*: every pair of nodes can communicate directly without intermediate hops.
Assumptions:

- Each node can simultaneously send and receive messages.

- Communication cost for a message of length n bytes between any two nodes is: $\mathrm{T(n)} = \alpha + \beta\, n$ where α is the latency (startup cost) and β is the factor based on bandwidth.
- If routing through multiple hops with M concurrent flows, communication cost becomes: $\mathrm{T(n)} = \alpha + M\,\beta\, n$
- For large process counts (e.g., 10,000 processes), storing full routing tables ($10^4 \times 10^4$) is impractical.

6 Communication Primitives

6.1 Broadcast

One-to-all communication. For instance, in a single call data from one single process (rank=0) is given to all other processes. Atomic broadcast ensures all-or-none delivery.

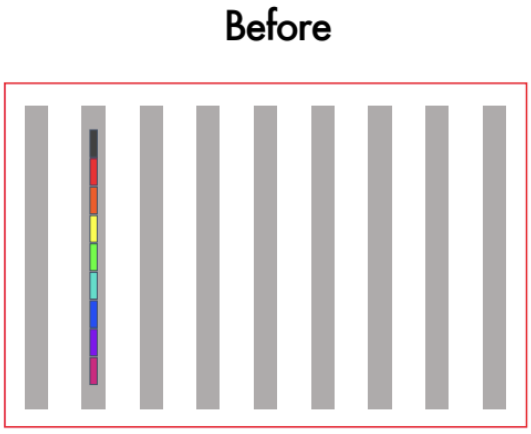

Figure 6: *
(a) Before

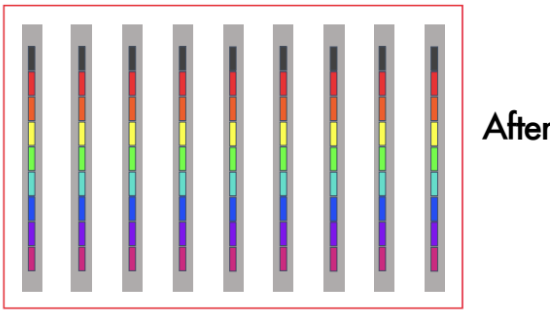

Figure 7: *
(b) After

Figure 8: Broadcast before and after

6.2 Reduce (to-one)

All-to-one communication. Each process contributes data, and applies associative function (e.g., sum, max) and combines them.

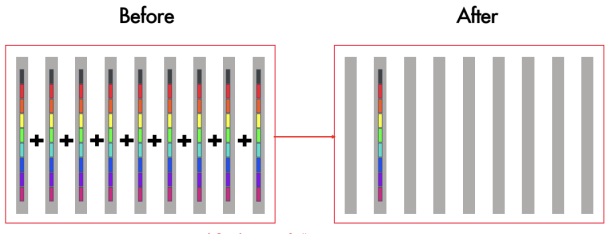

Figure 9: Reduce(to-one)

6.3 Broadcast/Reduce (to-one)

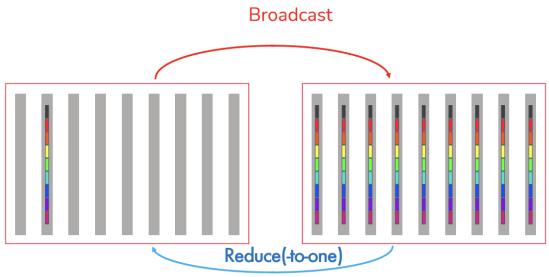

Figure 10: Broadcast/Reduce (to-one)

6.4 Scatter

One-to-all partitioned communication. Each root splits an array into different chunks and sends each chunk to different process. The root keeps its own chunk.

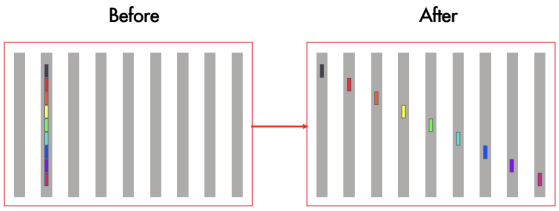

Figure 11: Scatter

6.5 Gather

All-to-one without reduction function. Each process sends its local chunk to the root, which concatenates them.

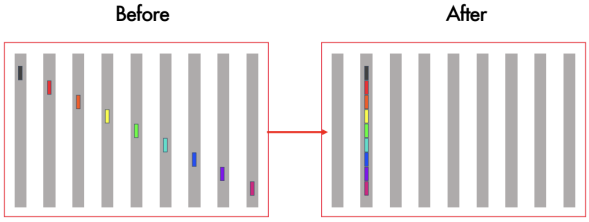

Figure 12: Gather

6.6 Scatter/Gather

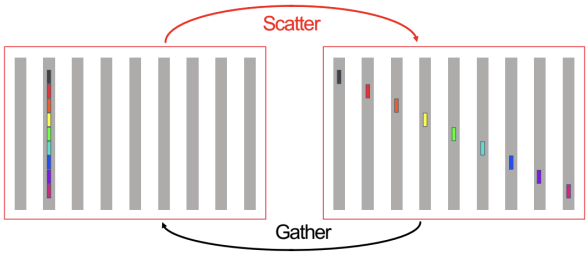

Figure 13: Scatter/Gather

6.7 All-Gather

Gather followed by broadcast, Every process ends up with the concatenation of all local chunks.

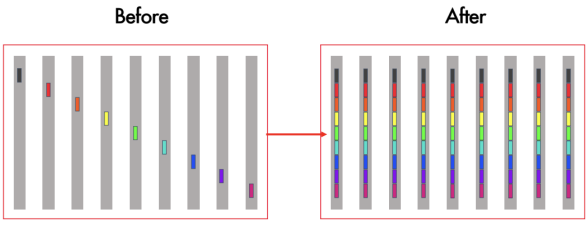

Figure 14: All-Gather

6.8 Reduce-Scatter

Opposite of all-gather. Data is reduced across all processes, then the result is partitioned and scattered so each process gets a segment.

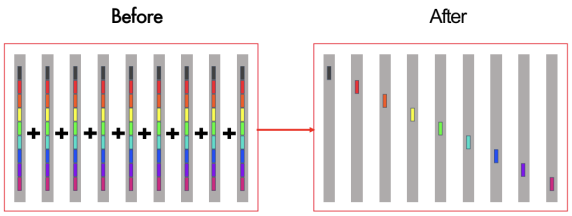

Figure 15: Reduce-Scatter

6.9 All-Gather/Reduce-Scatter

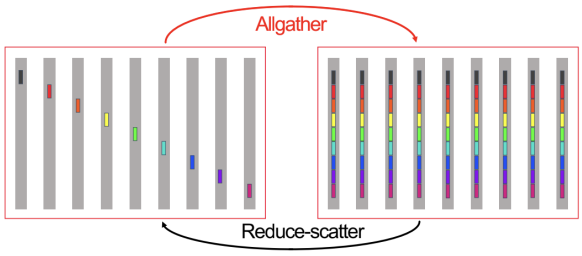

Figure 16: All-Gather/Reduce-Scatter

6.10 All-Reduce

Reduce followed by broadcast function. All processes end up with the reduced result of all inputs.

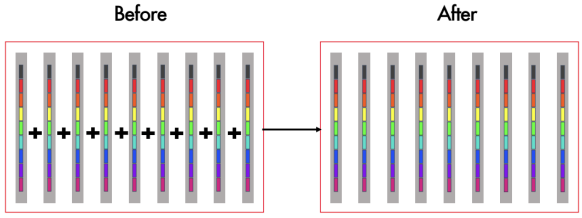

Figure 17: All-Reduce

7 Two Family of Mainstream Algorithms/Implementation

7.1 Minimal Spanning Tree (MST) Broadcast

- Edge weights represent communication costs between nodes.
- MST ensures minimum total latency but not useful for large messages.

7.2 Ring Algorithm

- Useful for large messages and gives emphasize to bandwidth utilization..

General principle:

1. At step i, divide the set of processes into two halves of size $\lceil P/2 \rceil$ and $\lfloor P/2 \rfloor$.
2. Each process in the half that already has data pairs with a unique process in the other half and sends (or receives) data.

3. Recursively apply within each half until all processes have the data.

This takes $\lceil \log_2 P \rceil$ steps, each costing $(\alpha + \beta\, n)$

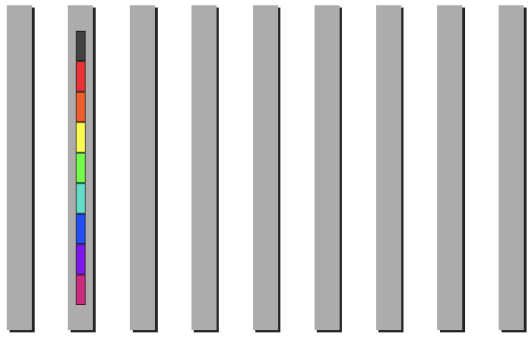

Figure 18: Step 1

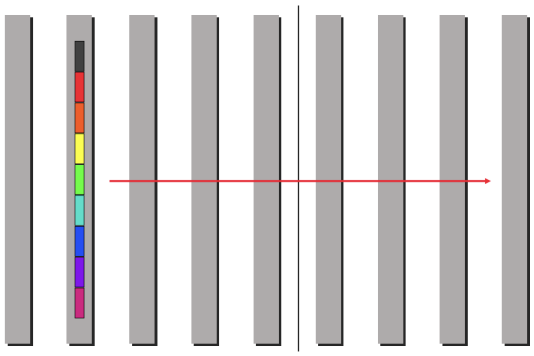

Figure 19: Step 2

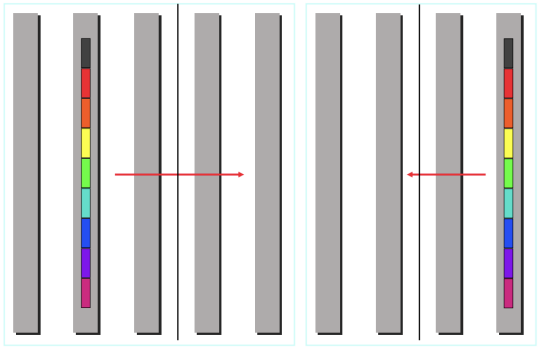

Figure 20: Step 3

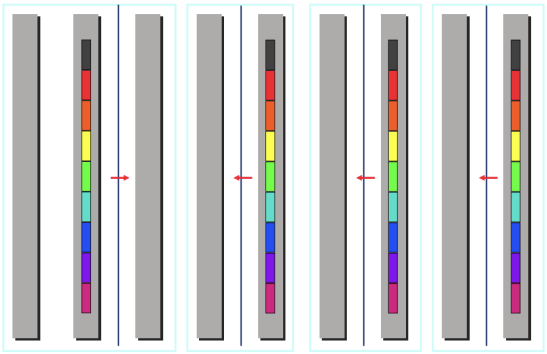

Figure 21: Step 4

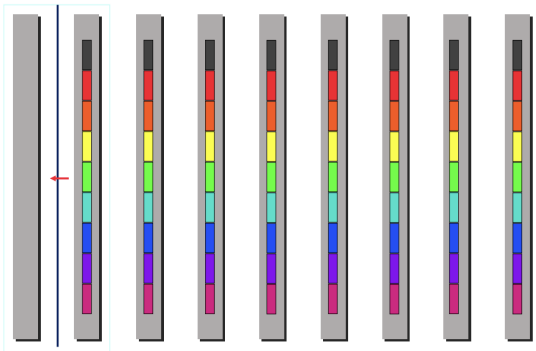

Figure 22: Step 5

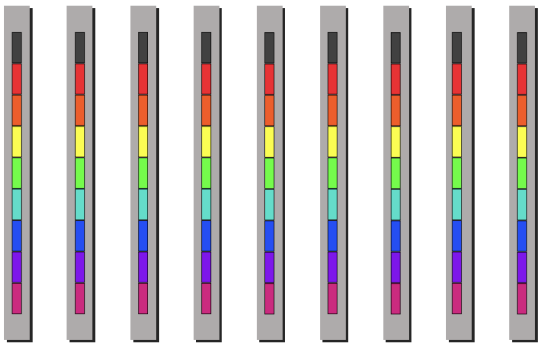

Figure 23: Step 6

Figure 24: Ring-algorithm steps