**Faculty of Information Engineering, Informatics and Statistics**
**Department of Computer Science**

# Big Data Computing

**Author:**
Simone Lidonnici

3 october 2025

# Contents

# 1 Introduction

With **Big Data** we refer to an actual phenomenon, defined by five properties called **5V**:

- **Value**: extracting knowledge from data is valuable.
- **Volume**: large amount of data.
- **Variety**: different format of data.
- **Velocity**: the speed at which the data are generated is very high.
- **Veracity**: reliability of the data used.

To do a computation on a lot of data at a high speed (like a google search), there are different problems that occurs:

- Disks are not large enough.
- Disks are not fast enough.
- CPUs are not fast enough.

## 1.1 Data center

To do this type of computations **data center** are used, which can upgrade their performances in two way:

- **Scale up**: buying new and more powerful components. This is a problem because the increase in the velocity required is faster then the performance increase.
- **Scale out**: buying more components and interconnect them to work in parallel.

A data center is composed by a series of rack interconnected between them with a private network. Every rack has servers inside them that contain CPUs, GPUs and memory. A server is considered a node in the network.
The major problem with the data centers is the bottleneck caused by the network. The network can't send all the data required for the computation in a fast way and this slow down the computation. Also, the increase in GPU power (that does the major part of computation in a data center) is much faster than the increase in network speed durind the years.

### 1.1.1 Reliability and Programmability problems

Another problem is related to **reliability**, so the probability that a server, disk or network component fails. Even if the probability of a single component is very low (one fail every 10 years) a large data center with thousand of components will have failures very frequently. This mean that checkpoint must be used, and every failure reset to the last one.

Last problem is the **programmability**, so how to write an efficient parallel program being aware of all the components. Things that have to be taken in account are how to store data, how to send data efficiently, what to execute on CPU or GPU and many other.

# 2 Distributed Deep Learning

**Deep Learning** performs better with more data and parameters, but also require more computation.
The training of a Distributed Neural Network (DNN) is done by doing many iteration of 3 staps:

1. **Forward propagation (Compute the function)**: apply the model to input and produce a prediction.
2. **Backward propagation (Compute the gradient)**: calculate the error and find the parameters that contribute more to the error to correct them.
3. **Parameters update (Use the gradient)**: use the loss value to update the model parameters.

## 2.1 Parallelize a DNN

There are 3 ways to parallelize a DNN:

- Data parallelism: divide the training dataset in batches and compute the model o each batch in parallel. After each iteration we synchronize the parameters of the different models by doing a gradient aggregation. This is done by doing an Allreduce on the weights vector across the GPUs.
- Pipeline parallelism: divide the model by the layers, assigning each layer to a different GPU. In the forward phase the first GPU output will be the input of the second GPU and so on. In the backward phase is the opposite. The problem is that the last GPUs have a lot of idle time during the forward phase and the first GPUs in the backward phase. This idle time is called bubble. To reduce this bubble we can divide the batch of data in micro-batches and do the forward and backward computation on the micro-batches. There are also other complex methods to reduce the bubble.
- Operator (or Tensor) parallelism: if all the parameters of a layer do not fit in a single GPU wehave to split the layer across multiple GPUs. An example could be splitting the matrices that we have to multiply (training require a lot of matrix multiplication) and using an Allreduce to sum the partial matrices. This requires a lot of communication.

In large models all three technique are usedat the same time.

## 2.2 Reducing communication overhead

To reduce the communication overhead we can compute and communicate in the same time, sending the gradients of every layer as soon as they are computed, so the computation of the gradient of the previous layer can be done during the communication of the gradient of the successive layer.
Another way is to compress the gradient and reducing his data volume, this can be done in three ways to have a **lossy compression**:

- Quantization: reduce the bitwidth of the elements (for example for float32 to float16). Some models converge even with only one bit per weight (knowing only if is positive or negative).
- Sparsification: sending only the larger $k$ values or by setting a threshold and sending only the weights bigger than it.
- Low rank: decomposes gradient into lower-rank matrices.

Lossy compression training converges with the same asymptotic rate, but in real life it requires more iterations.
Asynchronous sytems could be also used to speed up the process and overcoming the gradient aggregation that acts as a barrier. This can be done with different techniques, like aggregating after not one but many iterations or not waiting the slowest GPUs.

## 2.3 Collective operations used in DNN

In DNN different collective operations are used:

- Allreduce for gradient aggregation in data parallelism.
- Allgather and Reduce-Scatter in sharded data parallelism.
- Allgather and Allreduce for matrix multiply in operator parallelism.
- Alltoall in expert parallelism.

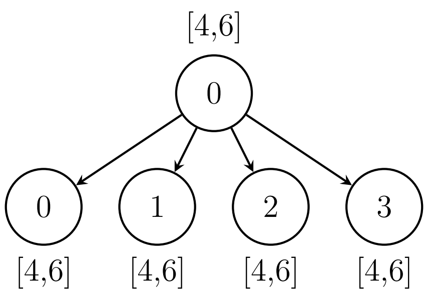
The cost model can be expressed with this formula:

$$T = k\alpha + \delta n\beta$$

In which $\alpha$ is the latency, a fixed cost, $\beta$ is the cost per byte sent, $n$ are the byte sent and $\delta$ is the fraction of data sent in respect to $n$, considered as the maximum between the input and the output.

### 2.3.1 Broadcast

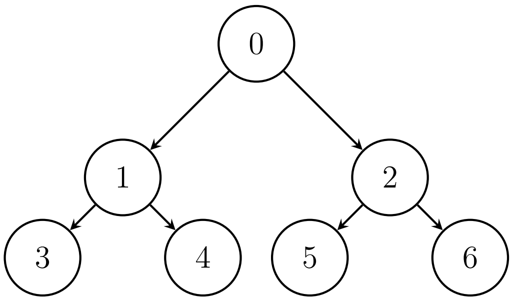
A simpler collective operation is the **Broadcast**, in which data from a process are copied to all the other processes.



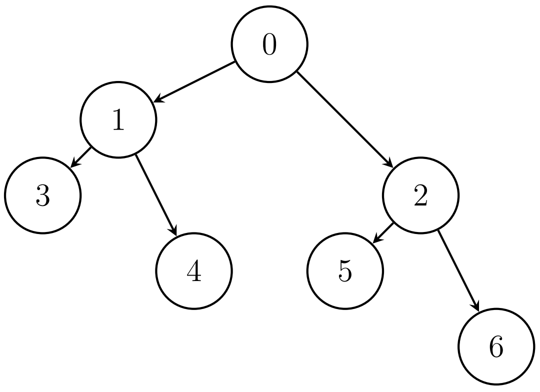
The basic algorithm to compute the Broadcast is the **chain algorithm**, in which at every step the last process to have recieved the data sends them to the successive. So, in step 0 the process 0 (who has the data at the start) sends them to the process 1, in te next step 1 will send them to process 2 and so on. The cost of this algorithm is:

$$T = (p - 1)(\alpha + n\beta)$$

Another algorithm involves the binary trees, in which at each step the processes that have received the data at the last step send all the data to two processes that don't have yet.

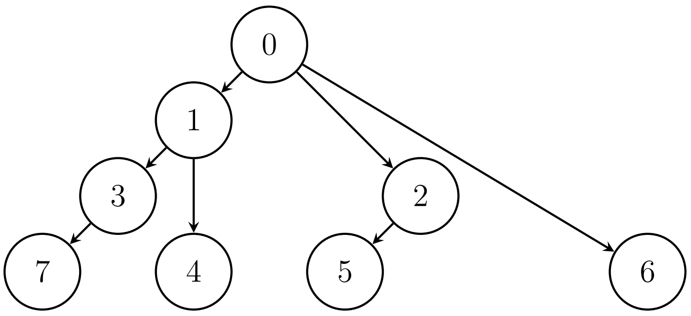


With concurrent communication:
$T = (\log(p + 1) - 1)(\alpha + n\beta)$



Without concurrent communication:
$T = 2(\log(p + 1) - 1)(\alpha + n\beta)$

Exists also an algorithm with binomial trees, in which a process doesn't stop sending after two steps but continue to send at every step.



The cost of this algorithm is:

$$T = \log p(\alpha + n\beta)$$

The Broadcast can also be done as a Scatter (takes a vector from a process and divide it evenly between all processes) plus a Allgather. The cost of the Scatter, using the binomial trees, is similar to the Broadcast algorithm, with the difference that the data sent halves every step, so for a step $k$ the data sent will be $\frac{n}{2^k}$. The cost of the Scatter is:

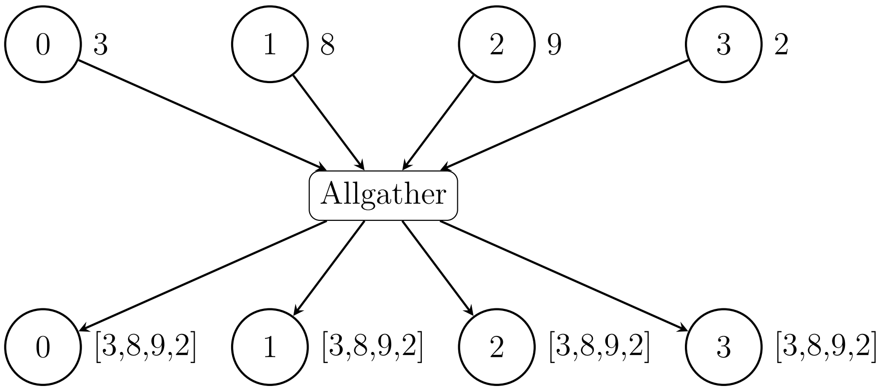
$$T = (\log p)\alpha + n\beta$$

For the Allgather we use the Recursive doubling algorithm explained in Section 2.3.2.
The total cost of this approach for the Broadcast is:

$$T = \underbrace{(\log p)\alpha + n\beta}_{\text{Scatter}} + \underbrace{(\log p)\alpha + n\beta}_{\text{Allgather}} = 2[(\log p)\alpha + n\beta]$$

### 2.3.2 Allgather

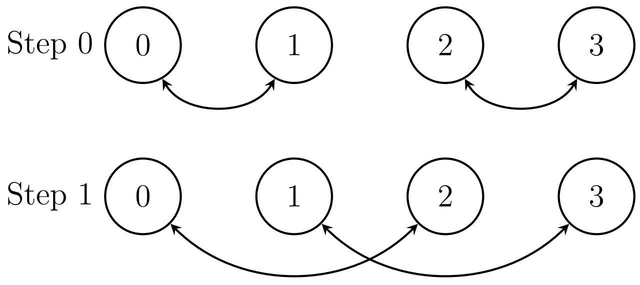
The **Allgather** is a collective operation that takes data from every process, combine them in a vector and copies the resulting vector to al the processes.



The basic algorithm to compute the Allgather consists in $p-1$ phases in which every process sends his data to another process. The cost is:

$$T = p\alpha + n\beta$$

Another simple algorithm is the **Ring** algorithm, which use the same structure of the basic algorithm, but instead of sending the data to a different process every step, a process $i$ will always send data to process $i+1$. The cost of the algorithm remain the same, but the difference is that the every link in the system is used by only one communication at a time.
The **Recursive doubling (Butterfly)** algorithm has a logaritmic number of phases and consist in exchanging data in a bidirectional way between process with distance that doubles at every step. In a step $k$, a process $i$ will exchange all the data with a process at distance $2^k$.

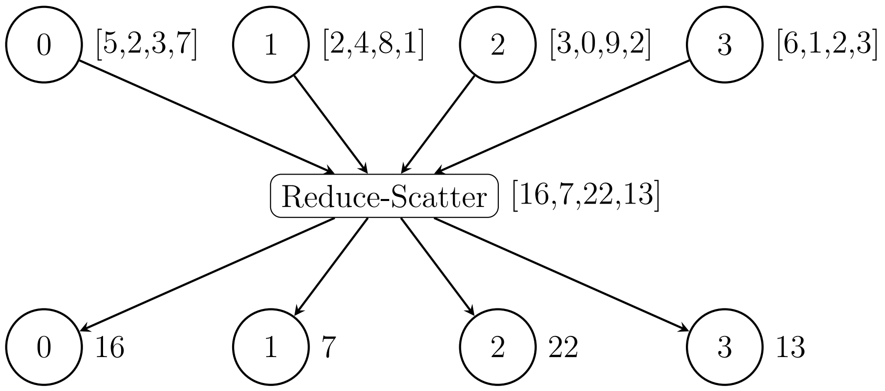


The cost is:

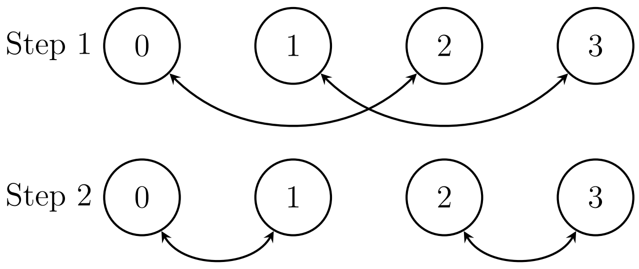
$$T = (\log p)\alpha + n\beta$$

### 2.3.3 Reduce-Scatter

The **Reduce-Scatter** is a collective operation that takes a vector of data from every process and execute an operation on them. After calculating the resulting vector, it is splitted evenly between the processes, following the indexes.



The basic and Ring algorithm for the Reduce-Scatter are the same algorithm of the Allgather but in reversing order of communication.
Instead of Recursive doubling, for Reduce-Scatter we use the **Recursive halving (Butterfly)** algorithm, which works in a similar way but with the distance that halves instead of doubling. Also the data exchanged is more in the first stage and halves every stage.

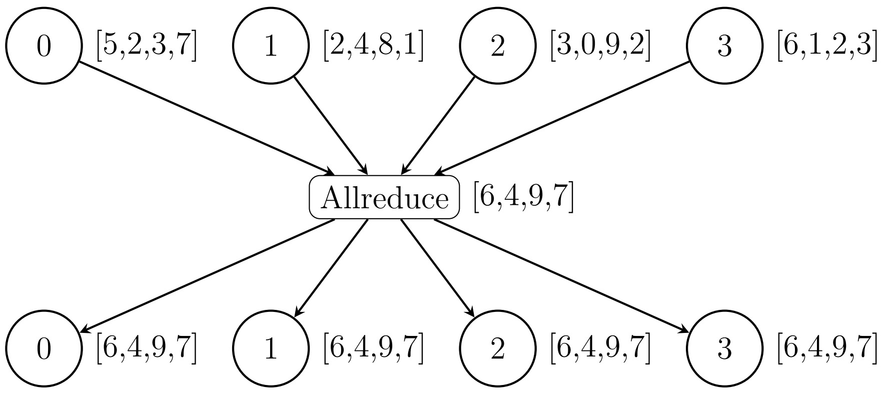


The cost is equal to the Recursive doubling algorithm of the Allgather:

$$T = (\log p)\alpha + n\beta$$

### 2.3.4 Allreduce

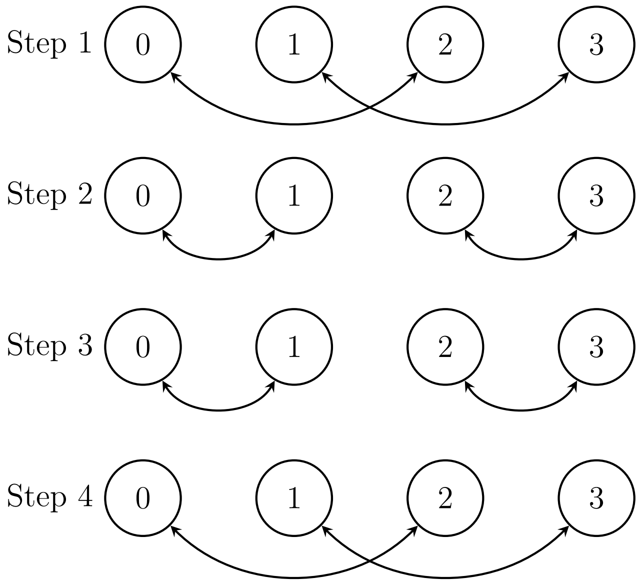
The **Allreduce** is a collective communication that takes a vector from every process, execute an operation on them and the resulting vector is copied in every process.



The Ring algorithm is done by executing a Reduce-Scatter and an Allgather with Ring algorithm, so the cost will be the sum of the two:

$$T = 2(p\alpha + n\beta)$$

The **Rabenseifner (Butterfly)** algorithm use a Reduce-Scatter with Recursive halving algorithm followed by an Allgather with Recursive doubling algorithm.



Also in this case the cost is the sum of the two algorithms:

$$T = 2[(\log p)\alpha + n\beta]$$

There is an algorithm that isn't the sum of a Reduce-Scatter and Allgather, the **Recursive doubling** algorithm. The algorithm is similar to the one to execute the Allgather but every step the processes send all the vector. The cost is:

$$T = (\log p)(\alpha + n\beta)$$

### 2.3.5 All-to-All