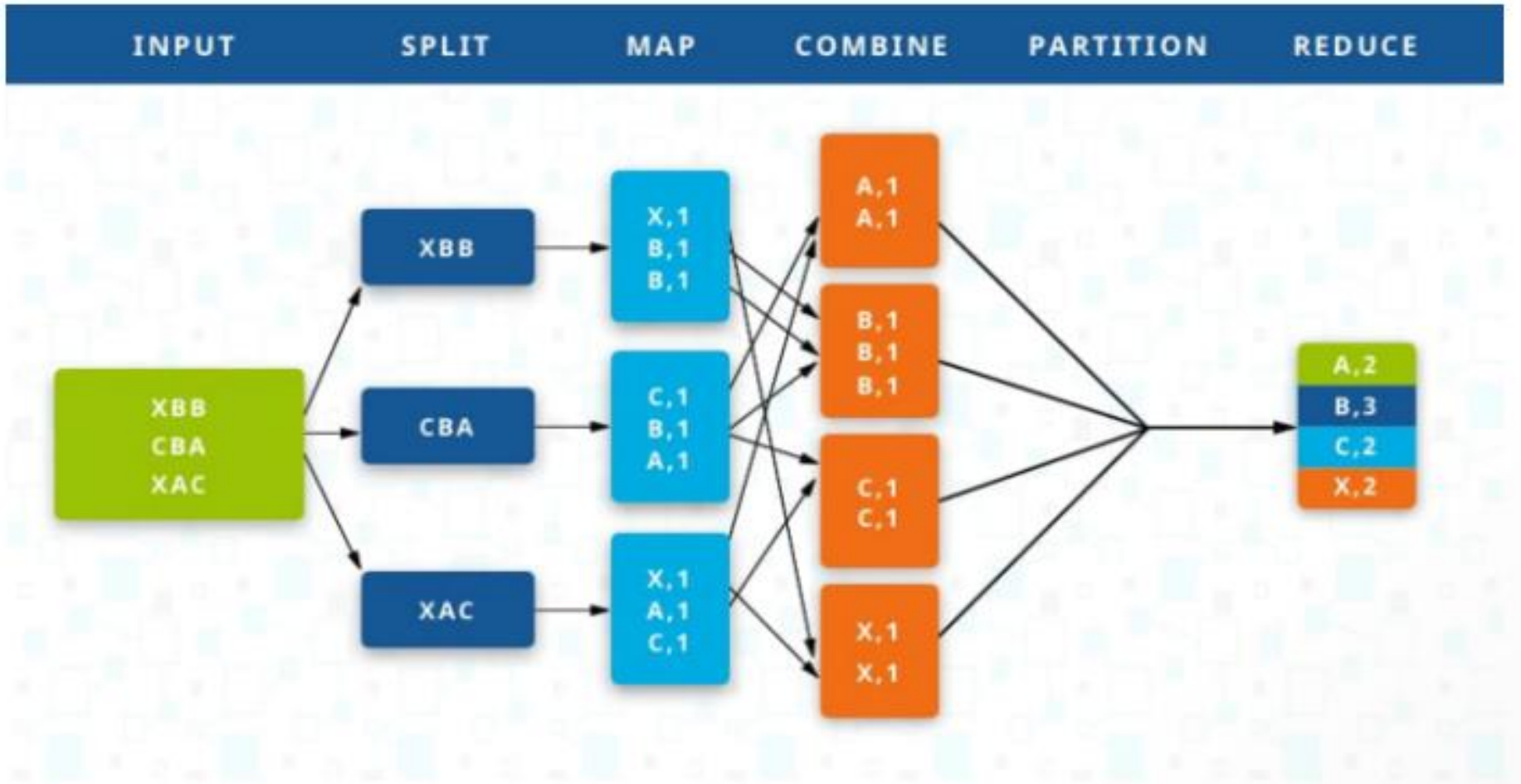
How MapReduce Works

Instructor: Hassan Jahangir

Week #03

- At the crux of MapReduce are two functions: Map and Reduce. They are sequenced one after the other.
- The **Map** function takes input from the disk as <key,value> pairs, processes them, and produces another set of intermediate <key,value> pairs as output.
- The **Reduce** function also takes inputs as <key,value> pairs, and produces <key,value> pairs as output.


INPUT
SPLIT
MAP
COMBINE
PARTITION
REDUCE
XBB
CBA
XAC
XBB
CBA
XAC
X,1
B,1
B,1
C,1
B,1
A,1
X,1
A,1
C,1
A,1
A,1
B,1
B,1
B,1
C,1
C,1
X,1
X,1
A,2
B,3
C,2
X,2

- The types of keys and values differ based on the use case. All inputs and outputs are stored in the HDFS. While the map is a mandatory step to filter and sort the initial data, the reduce function is optional.
- <k1, v1> -> Map() -> list(<k2, v2>)
  <k2, list(v2)> -> Reduce() -> list(<k3, v3>)
- Mappers and Reducers are the Hadoop servers that run the Map and Reduce functions respectively. It doesn’t matter if these are the same or different servers.
-

Map

- The input data is first split into smaller blocks. Each block is then assigned to a mapper for processing.
- For example, if a file has 100 records to be processed, 100 mappers can run together to process one record each. Or maybe 50 mappers can run together to process two records each. The Hadoop framework decides how many mappers to use, based on the size of the data to be processed and the memory block available on each mapper server.

Reduce

- After all the mappers complete processing, the framework shuffles and sorts the results before passing them on to the reducers. A reducer cannot start while a mapper is still in progress. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key.

Combine and Partition

- **Combine** is an optional process. The combiner is a reducer that runs individually on each mapper server. It reduces the data on each mapper further to a simplified form before passing it downstream.
- This makes shuffling and sorting easier as there is less data to work with. Often, the combiner class is set to the reducer class itself, due to the cumulative and associative functions in the reduce function. However, if needed, the combiner can be a separate class as well.
- **Partition** is the process that translates the <key, value> pairs resulting from mappers to another set of <key, value> pairs to feed into the reducer. It decides how the data has to be presented to the reducer and also assigns it to a particular reducer.
- The default partitioner determines the hash value for the key, resulting from the mapper, and assigns a partition based on this hash value. There are as many partitions as there are reducers. So, once the partitioning is complete, the data from each partition is sent to a specific reducer.

A MapReduce Example

- Consider an ecommerce system that receives a million requests every day to process payments. There may be several exceptions thrown during these requests such as "payment declined by a payment gateway," "out of inventory," and "invalid address." A developer wants to analyze last four days' logs to understand which exception is thrown how many times.

Example Use Case

- The objective is to isolate use cases that are most prone to errors, and to take appropriate action. For example, if the same payment gateway is frequently throwing an exception, is it because of an unreliable service or a badly written interface? If the "out of inventory" exception is thrown often, does it mean the inventory calculation service has to be improved, or does the inventory stocks need to be increased for certain products?
- The developer can ask relevant questions and determine the right course of action. To perform this analysis on logs that are bulky, with millions of records, MapReduce is an apt programming model. Multiple mappers can process these logs simultaneously: one mapper could process a day's log or a subset of it based on the log size and the memory block available for processing in the mapper server.

Map

- For simplification, let's assume that the Hadoop framework runs just four mappers. Mapper 1, Mapper 2, Mapper 3, and Mapper 4.
- The value input to the mapper is one record of the log file. The key could be a text string such as "file name + line number." The mapper, then, processes each record of the log file to produce key value pairs. Here, we will just use a filler for the value as '1.' The output from the mappers look like this:
- Mapper 1 -> <Exception A, 1>, <Exception B, 1>, <Exception A, 1>, <Exception C, 1>, <Exception A, 1>
  Mapper 2 -> <Exception B, 1>, <Exception B, 1>, <Exception A, 1>, <Exception A, 1>
  Mapper 3 -> <Exception A, 1>, <Exception C, 1>, <Exception A, 1>, <Exception B, 1>, <Exception A, 1>
  Mapper 4 -> <Exception B, 1>, <Exception C, 1>, <Exception C, 1>, <Exception A, 1>

- Assuming that there is a combiner running on each mapper—Combiner 1 … Combiner 4—that calculates the count of each exception (which is the same function as the reducer), the input to Combiner 1 will be:
- <Exception A, 1>, <Exception B, 1>, <Exception A, 1>, <Exception C, 1>, <Exception A, 1>

Combine

- The output of Combiner 1 will be:
- <Exception A, 3>, <Exception B, 1>, <Exception C, 1>
- The output from the other combiners will be:
- Combiner 2: <Exception A, 2> <Exception B, 2>
  Combiner 3: <Exception A, 3> <Exception B, 1> <Exception C, 1>
  Combiner 4: <Exception A, 1> <Exception B, 1> <Exception C, 2>

Partition

- After this, the partitioner allocates the data from the combiners to the reducers. The data is also sorted for the reducer.
- The input to the reducers will be as below:
- Reducer 1: <Exception A> {3,2,3,1}
  Reducer 2: <Exception B> {1,2,1,1}
  Reducer 3: <Exception C> {1,1,2}
- If there were no combiners involved, the input to the reducers will be as below:
- Reducer 1: <Exception A> {1,1,1,1,1,1,1,1,1}
  Reducer 2: <Exception B> {1,1,1,1,1}
  Reducer 3: <Exception C> {1,1,1,1}
- Here, the example is a simple one, but when there are terabytes of data involved, the combiner process' improvement to the bandwidth is significant.

Reduce

- Now, each reducer just calculates the total count of the exceptions as:
- Reducer 1: <Exception A, 9>
  Reducer 2: <Exception B, 5>
  Reducer 3: <Exception C, 4>
- The data shows that Exception A is thrown more often than others and requires more attention. When there are more than a few weeks' or months' of data to be processed together, the potential of the MapReduce program can be truly exploited.