1. **What is input split and how to configure mappers based on need?**

The data is divided logically for each individual mapper job. And these physically divided data are stored in the form of blocks. Each mapper job can split the data into any number of blocks.

The basic need of input splits is to divide the data into logical blocks so that each mapper can process the total set of data. And the number of input splits are equal to the number of mappings.

InputFormat.getSplits() function is responsible for input splits.

Suppose if we have 150MB of file size and the HDFS data block of 128MB, then there will be 2 input splits.

1. **HADOOP configuration files in detail.**

*hadoop-env.sh* 🡪 environment variables that are used to run hadoop.

*Core-site.xml* 🡪configuration settings for input/output settings that are common for HDFS and MapReduce

*hdfs-site.xml* 🡪 configuration settings for hdfs daemons name node, secondary name node and the data nodes.

*mapred-site.xml* 🡪 configuration settings for MapReduce daemons job tracker and the task tracker.

*Masters* 🡪 a list of machines in each line that run the secondary name node and Job trackers.

*Slaves* 🡪 a list of machines in each line that run data nodes and task trackers.

1. **What is block in hdfs?**

In HADOOP, hdfs splits the large file into small packets of data called as blocks. The blocks are the smallest unit of data in a file system. However we don’t have any control on the blocks such as block location. The name node decides all such factors.

The size of all blocks are same except the last block. The last block can be either of the same size or smaller. The default size of each block in HDFS is either 64MB and latest, it is 128MB. Consider an example if a file is of 550MB.

The hdfs divides this file size into smaller size of 128MB in each block and the last block is of smaller.

It is distributed into 5 blocks, out of which 4 blocks are of 128MB and the last block is of 48MB.

1. **What is HDFS?**

HDFS is a highly scalable file system. As per recent update, the HDFS can split into 6000 nodes and 120 petabyte.

HDFS supports parallel reading and processing of data. (Read, write, rename, append).

HDFS splits the large data into smaller blocks and that is how the MapReduce functions are implemented. HDFS replicates the data into smaller servers(input split) that are in the cluster and the mappings are done to each server and data is processed.

1. **What is MapReduce?**

MapReduce is the main functionality of the HDFS. Whatever the data is stored in the HDFS, that should be processed. So when the HDFS cluster has the huge data file, that is split into smaller blocks of data (process called input split) and each mapper is assigned to each block of data. So each mapper processes the data according to the category they are. Again these output after the mapping task taken as input by the reduce task and the data blocks undergo shuffling and are reduced into the smaller tuples. These output is again sent back to the Hadoop distributed file system storage.

1. **Different stages of MapReduce, shuffling, mapper output, partitioner and combiner.**

In MapReduce,a YARN application is called a job. The implementation of the application master is provided by the MapReduce framework is called MRAppMaster.

There are 2 phases in MapReduce function. In Map phase,several map tasks are executed and in Reduce phase,several reduce tasks are executed.

Map Phase

Master needs to start for the current job.

The steps in a Map phase are:

a map() implementation

a combiner implementation

The number of files inside the input directory is used for deciding the number of Map Tasks of a job.

Map Tasks:

The Application Master will launch one MapTask for each map split. Typically, there is a map split for each input file. If the input file is too big (bigger than the HDFS block size) then we have two or more map splits associated to the same input file.

MapTask Launch

The MapReduce Application Master asks to the Resource Manager for Containers needed by the Job: one MapTask container request for each MapTask (map split).The Application Master asks for a container located on the same Node Manager where the map split is stored (a map split may be stored on multiple nodes due to the HDFS replication factor).otherwise, a container located on a Node Manager in the same rack where the the map split is stored;otherwise, a container on any other Node Manager of the cluster.

INIT phase: we setup the Map Task

EXECUTION phase: for each (key, value) tuple inside the map split we run the map() function

SPILLING phase: the map output is stored in an in-memory buffer; when this buffer is almost full then we start (in parallel) the spilling phase to remove data from it.

SHUFFLE phase: at the end of the spilling phase, we merge all the map outputs and package them for the reduce phase

Reduce phase:

* The number of ReduceTasks for the job is decided by the configuration parameter mapreduce.job.reduces.

Partitioner:

The parition of an output tuple is the index of a partition. It is decided inside the Mapper.Context.write():

* It is stored as metadata in the circular buffer alongside the output tuple. The user can customize the partitioner by setting the configuration parameter mapreduce.job.partitioner.class.

Combiner:

If the user specifies a combiner then the SPILLING thread, before writing the tuples to the file (4), executes the combiner on the tuples contained in each partition.

* create an instance of the user Reducer.class (the one specified for the combiner!)
* create a Reducer.Context: the output will be stored on the local filesystem
* execute Reduce.run(): see Reduce Task description
* The combiner typically use the same implementation of the standard reduce() function and thus can be seen as a local reducer.