Yet Another Map Reduce

Project Description

Yet Another Map Reduce is one of the project titles that can be taken up as a part of the Big Data course - *UE20CS322*. You are expected to implement the core components of Hadoop's Map Reduce Framework as a part of this project.

Project Objectives and Outcomes

- You will gain a deeper understanding of how MapReduce jobs are executed parallely across multiple nodes.
- At the end of this project, you will be able to setup a multi-node configuration that can store input data across multiple nodes and run Map and Reduce jobs

Technologies / Languages to be used

- You are free to any language of your choice such as Python, Java or JavaScript for this project.
- Ensure that the language that you choose supports the functionalities expected in this project.
- You are allowed to use any external libraries or APIs if required.

Marks

- 10 Marks

Deadline

- Final code submission to GitHub should be done by 11:59PM on November 27th.
- Commits after the deadline will not be considered.
- The last commit before the deadline will be used for a plagiarism check.
- Rules of plagiarism check remain the same as the previous assignments.

Evaluation

- Last working week of the semester, i.e., November 28th to December 2nd.
- Evaluation will be done in class hours and consist of project demonstration and viva.
- All the team members are **required** to be present and participating.

Project Specification

High level overview of execution

- You are required to setup a multinode environment consisting of a master node and multiple worker nodes.
- You are also required to setup a client program that communicates with the nodes based on the types of operations requested by the user.
- The types of operations that expected for this project are:
  - WRITE: Given an input file, split it into multiple partitions and store it across multiple worker nodes.
  - READ: Given a file name, read the different partitions from different workers and display it to the user.
  - MAP-REDUCE - Given an input file, a mapper file and a reducer file, execute a MapReduce Job on the cluster.

To help you get started with the project, the sections below will provide a detailed description of the workflow to execute each of the above operations.

Cluster Configuration and Setup

The components of the server include the following:

- Client Node / Program
- Master Node
- Worker Node(s)
  - The number of workers is a configurable parameter and will be provided as input during run time.

Definition of a Node

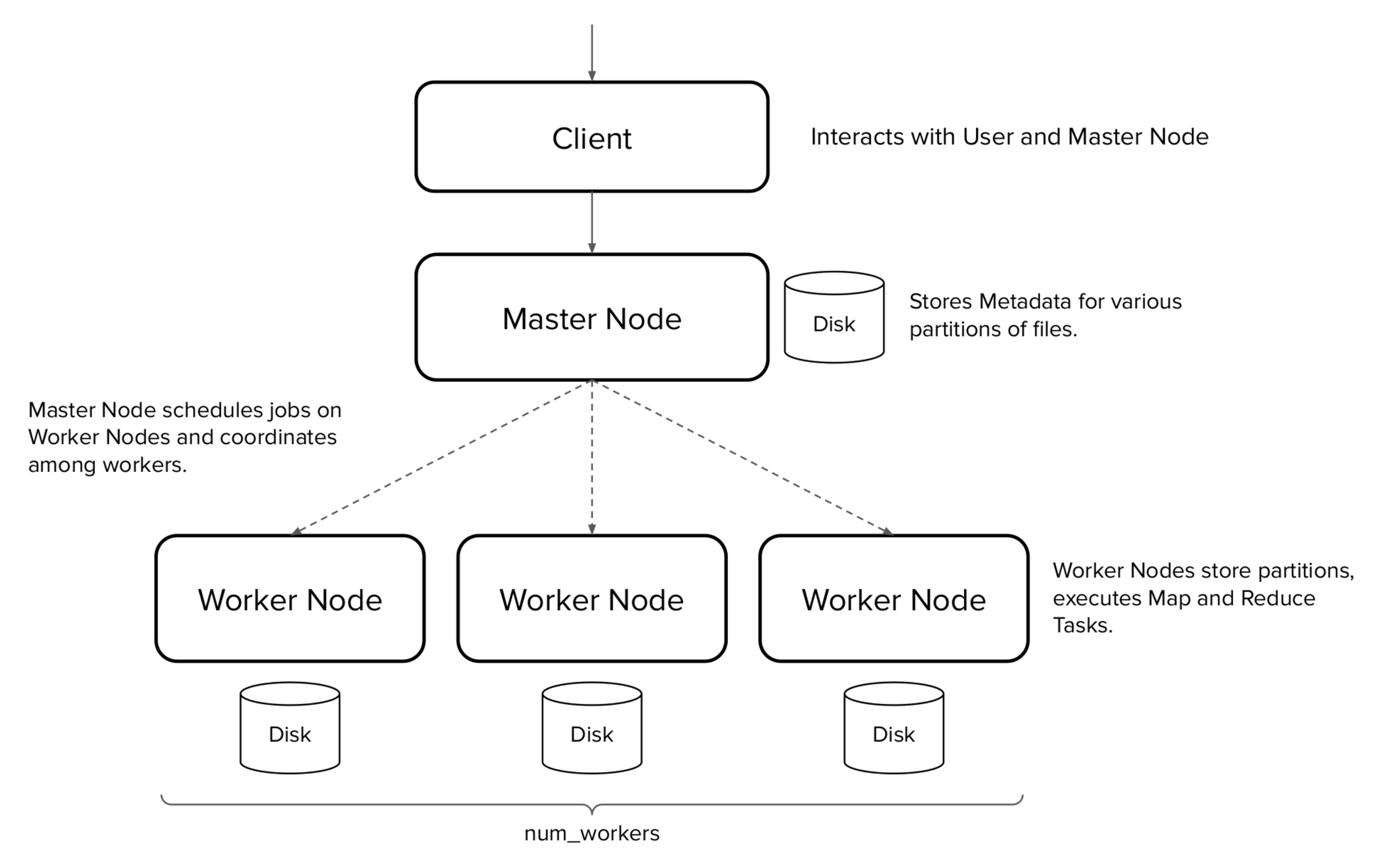
- You can choose to implement a node as a process, a virtual machine, a docker container or a seperate computer.
- Each node has access to a dedicated storage space on the filesystem.
- Each Node is expected to be running an HTTP server that has routes defined to communicate with other nodes in the system.
- Each node is represented by `<ip address>:<port>`
  - If you choose to run each node as a process on the same machine, the IP for all the nodes would be the same, so a node can be represented as `localhost:<port>`. Therefore, every node has to be assigned a unique port number.
- Implementing a node as a virtual machine, docker container or a seperate computer is an optional task.

Node wise responsibilites

This section is only an introduction to the responsibilites of each node; Further sections contain the responsibilities in more detail.

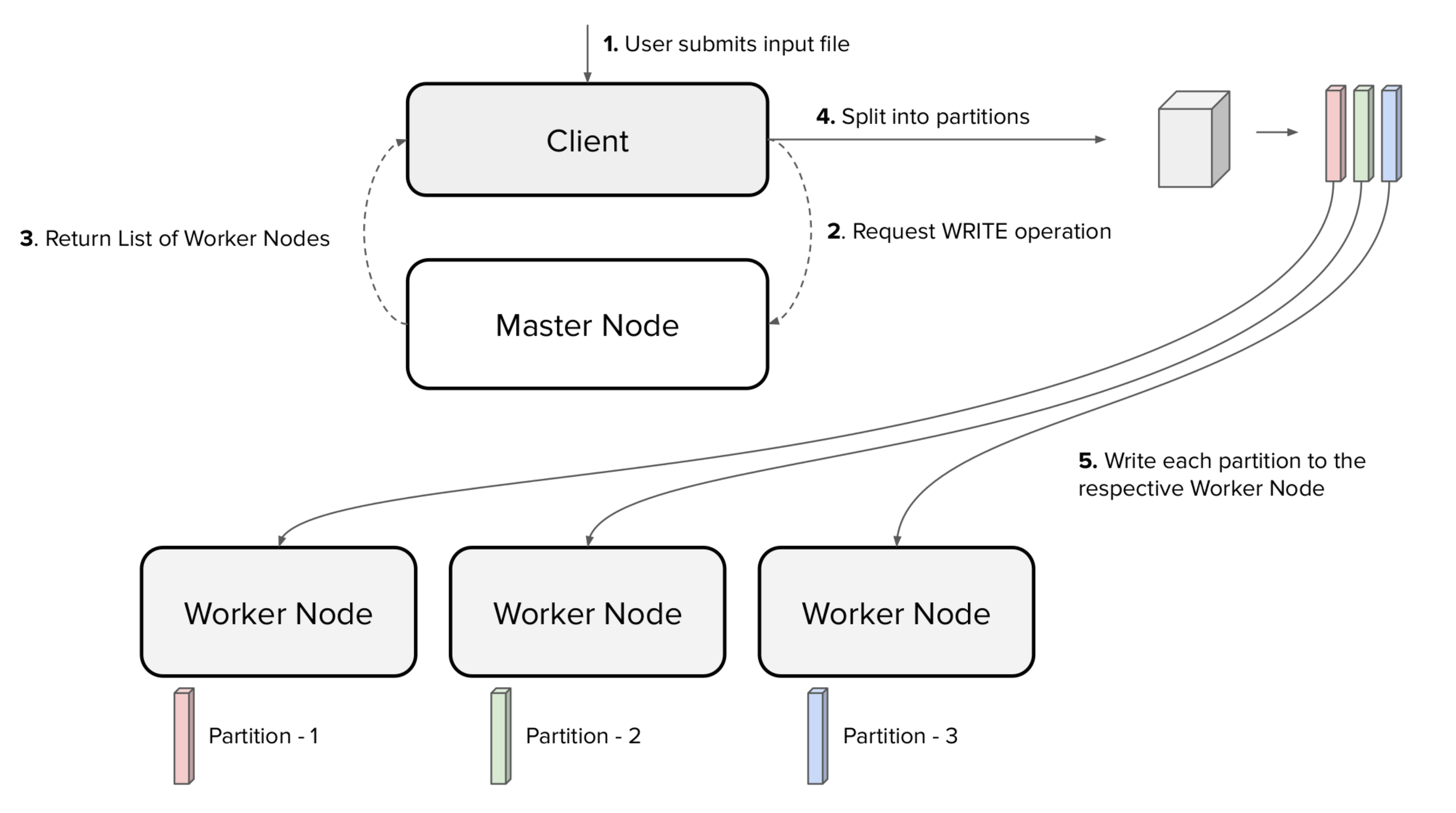
- Client Node
  - Accepting input for different operation from the user.
  - Interacting with the Master and the Worker nodes to complete user requests.
  - The client is the only process that the user interacts with.
- Master Node
  - Accepting operations from the client.
  - Scheduling Map and Reduce tasks on workers.
  - Co-ordinating between different workers to complete tasks.
  - Storing cluster wide metadata.
- Worker Node
  - Storing input data partitions.
  - Running Map and Reduce tasks on the stored partitions in **parallel**.

The pictorial representation of the setup is as follows:



The following sections will define the workflow for each of WRITE, READ and MAP-REDUCE operations.

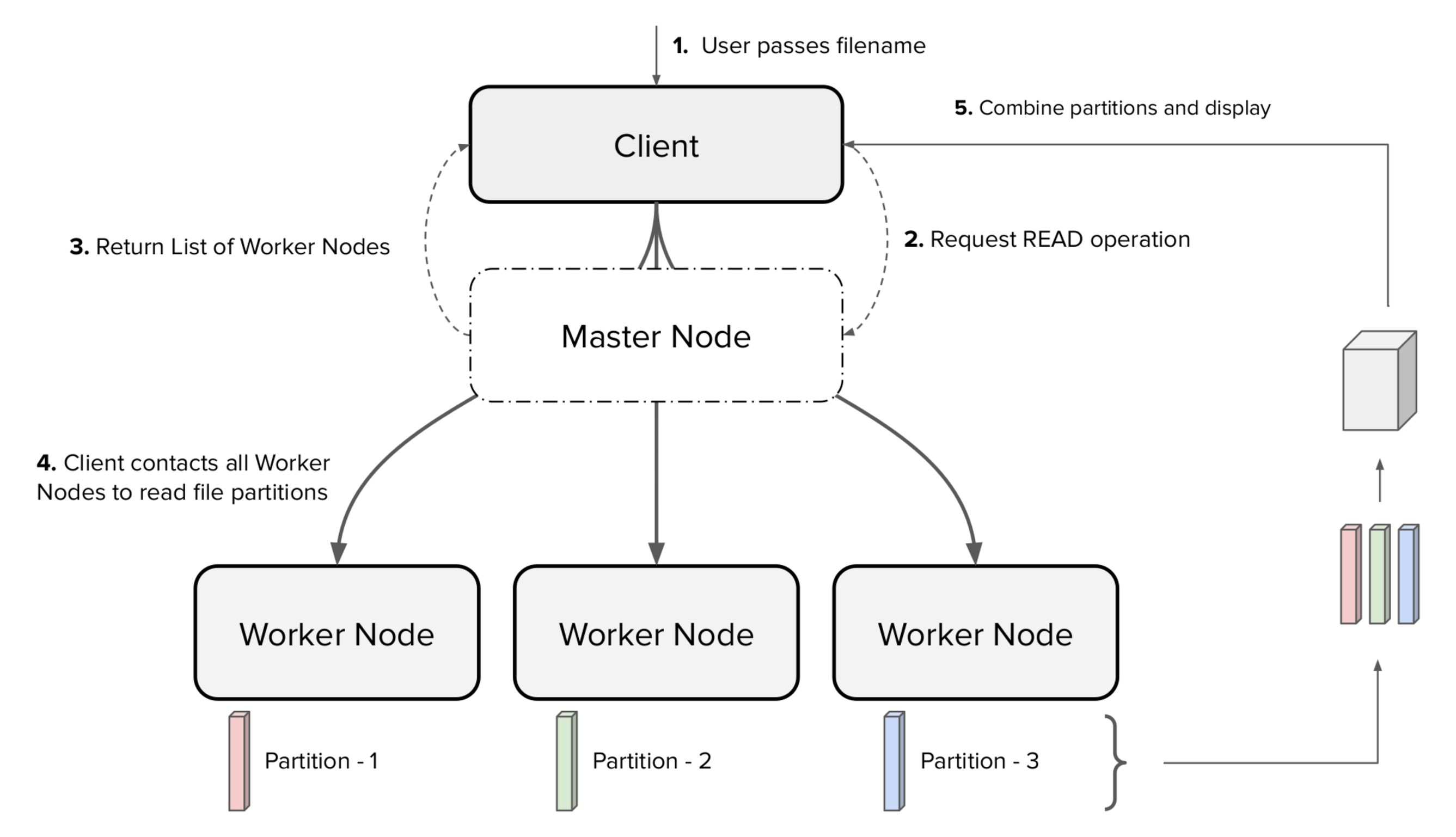
WRITE operation workflow



As shown in the image, the general workflow for a write operation is as follows:

- User passes the input file to the client program to be stored in the cluster.
- The client contacts the master node to schedule the WRITE operation.
- The master node returns a list of worker nodes that the client has to write the data to.
- Based on the list, the client program splits the input file into equally sized partitions and contacts the workers to store their respective partitions to the workers storage.
- After the client has successfully written the data to all the workers, client informs the user that the WRITE operation is successful.

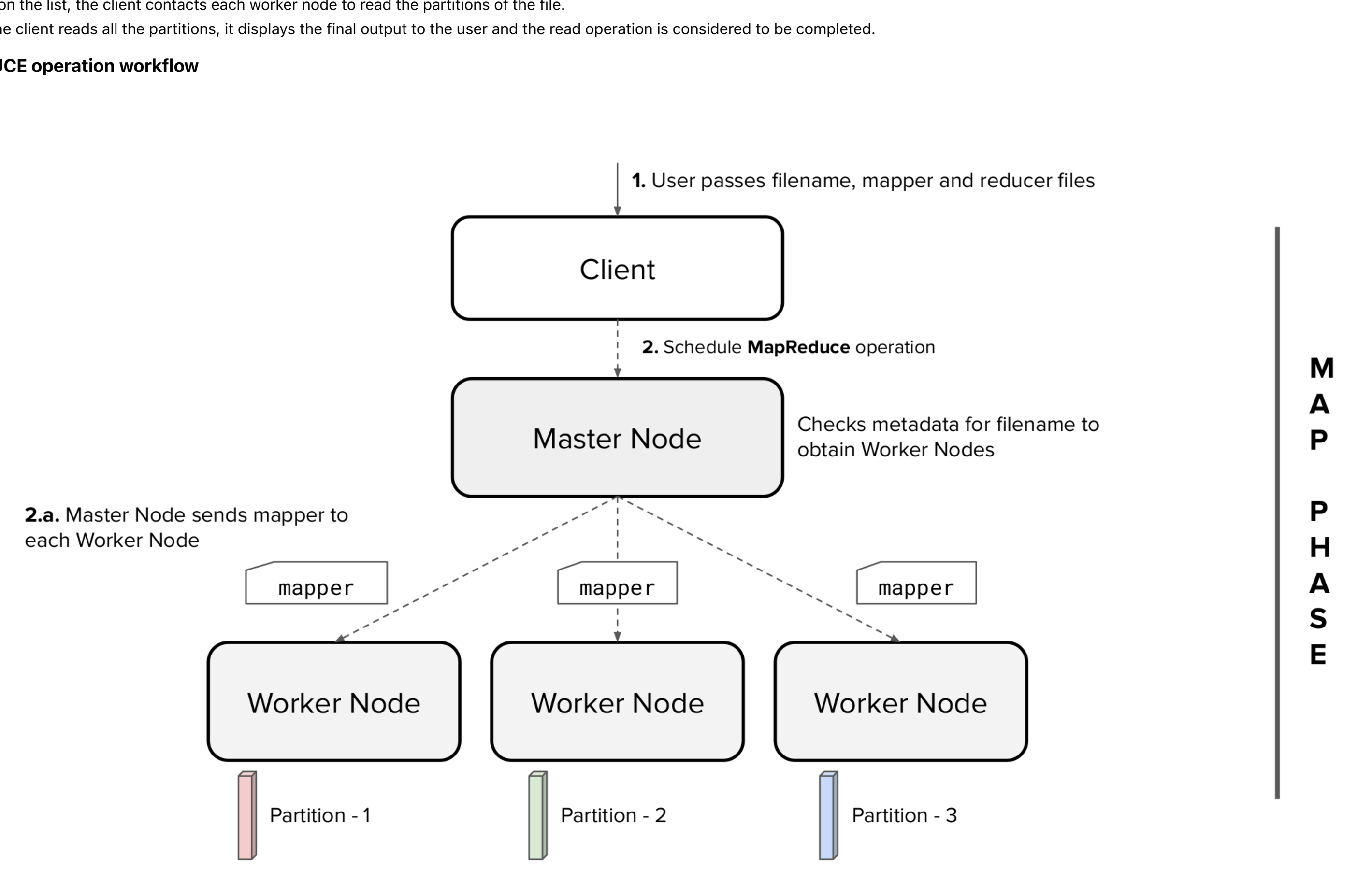
READ operation workflow

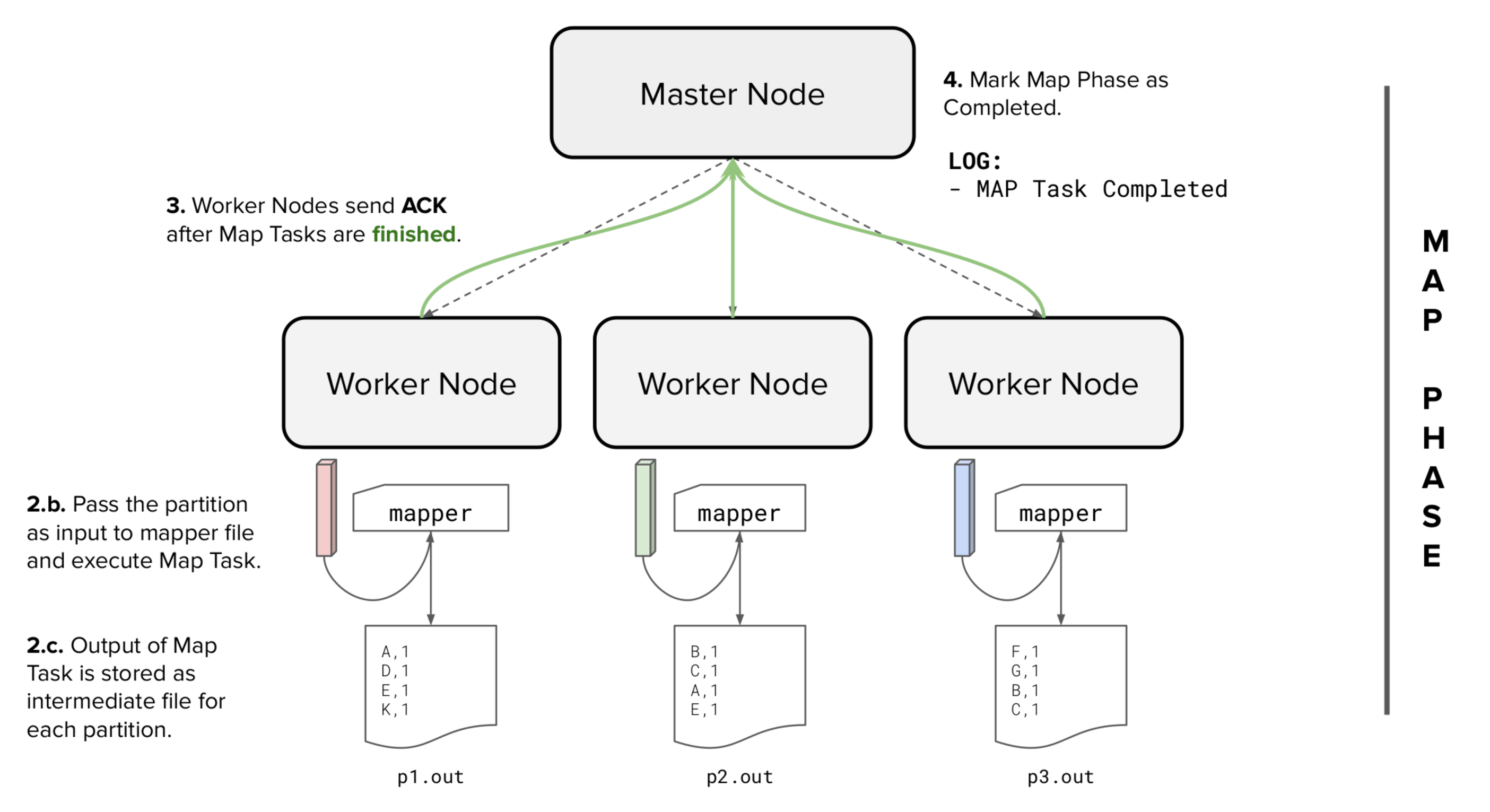


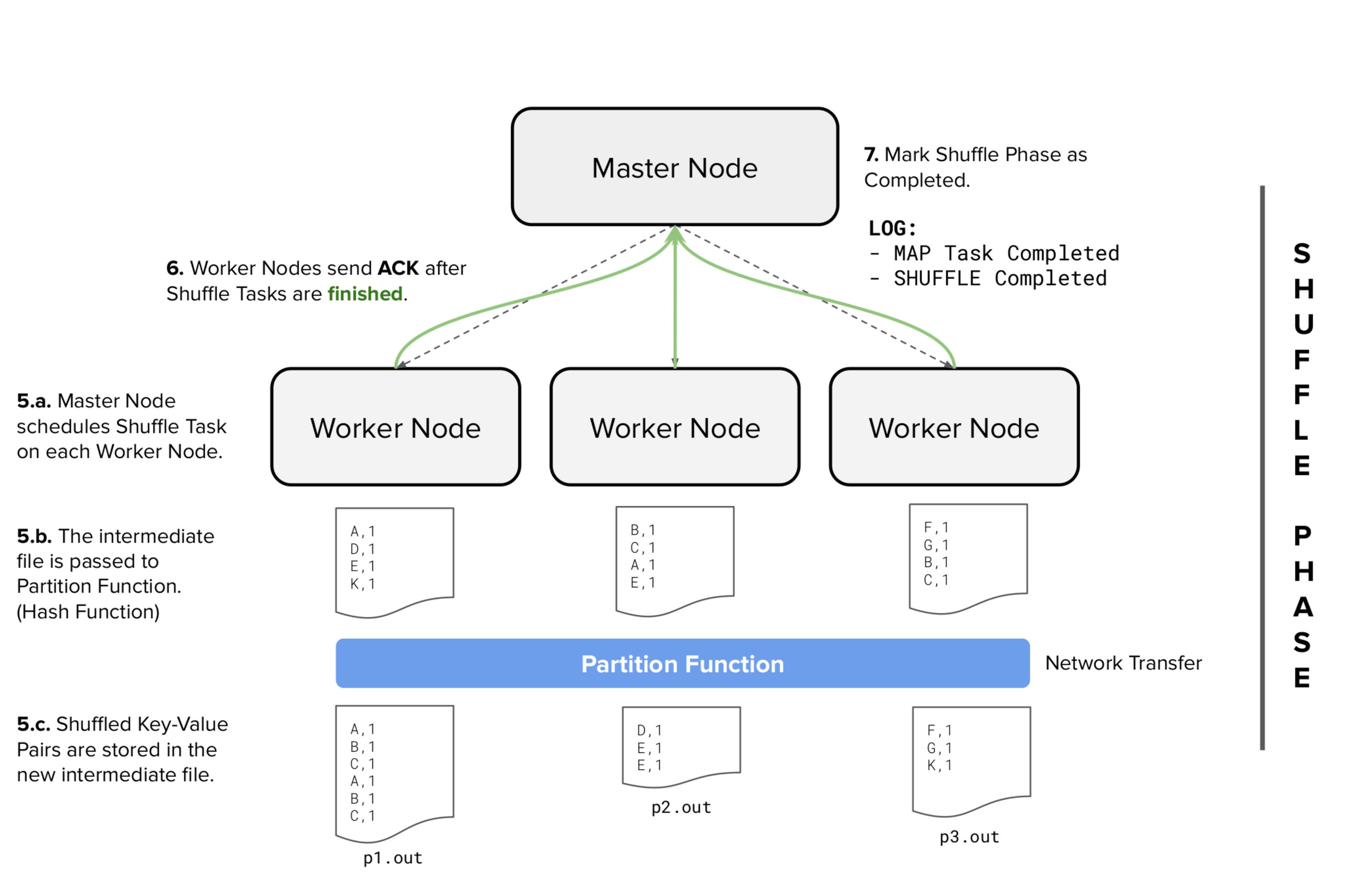
As shown in the image, the general workflow for a read operation is as follows:

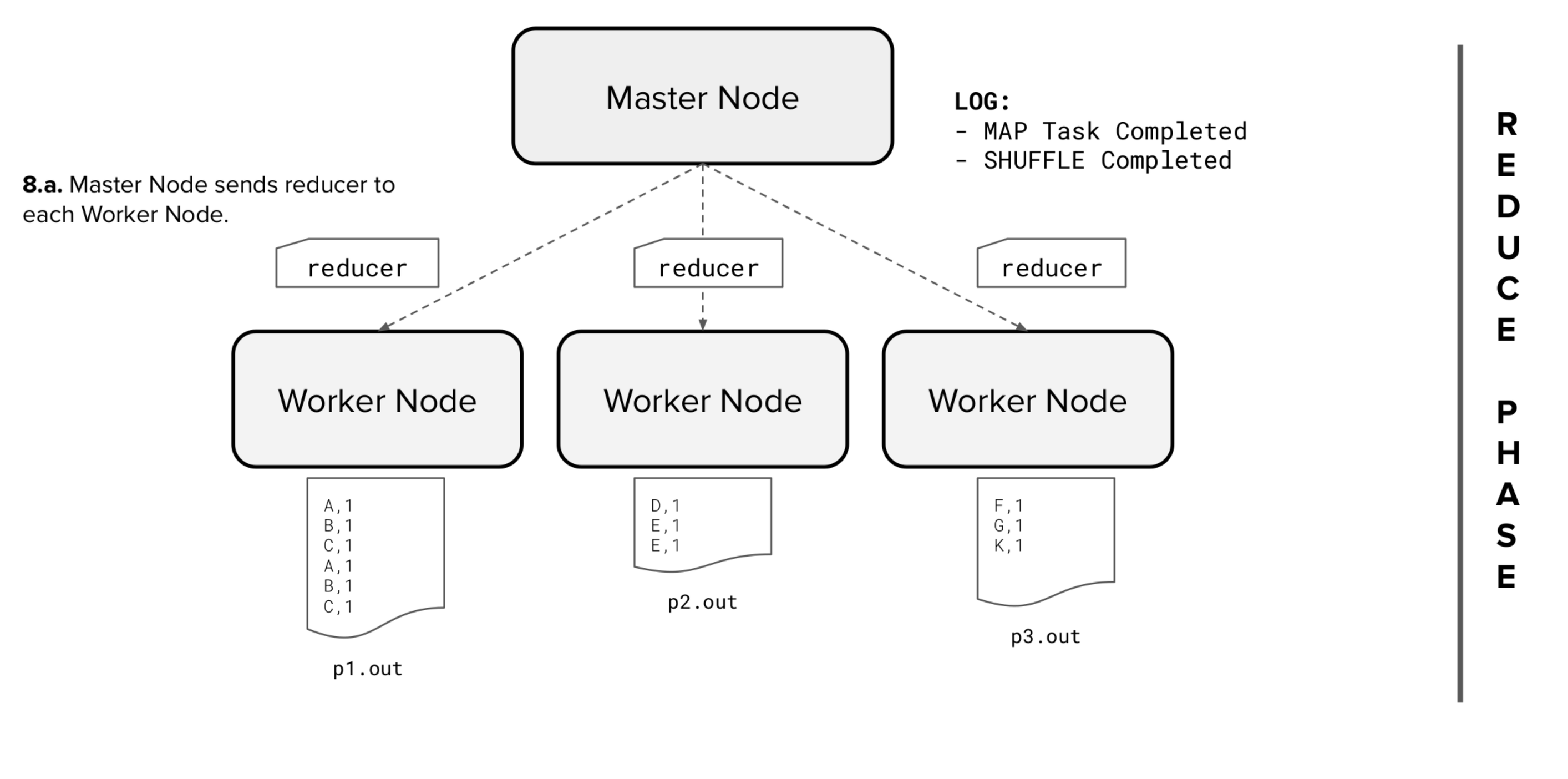
- User passes the name of the file that it wants to read from the cluster.
- The client contacts the master node to schedule the READ operation.
- The master node returns a list of worker nodes that contain the data for the file that the client wants to read.
- Based on the list, the client contacts each worker node to read the partitions of the file.
- After the client reads all the partitions, it displays the final output to the user and the read operation is considered to be completed.

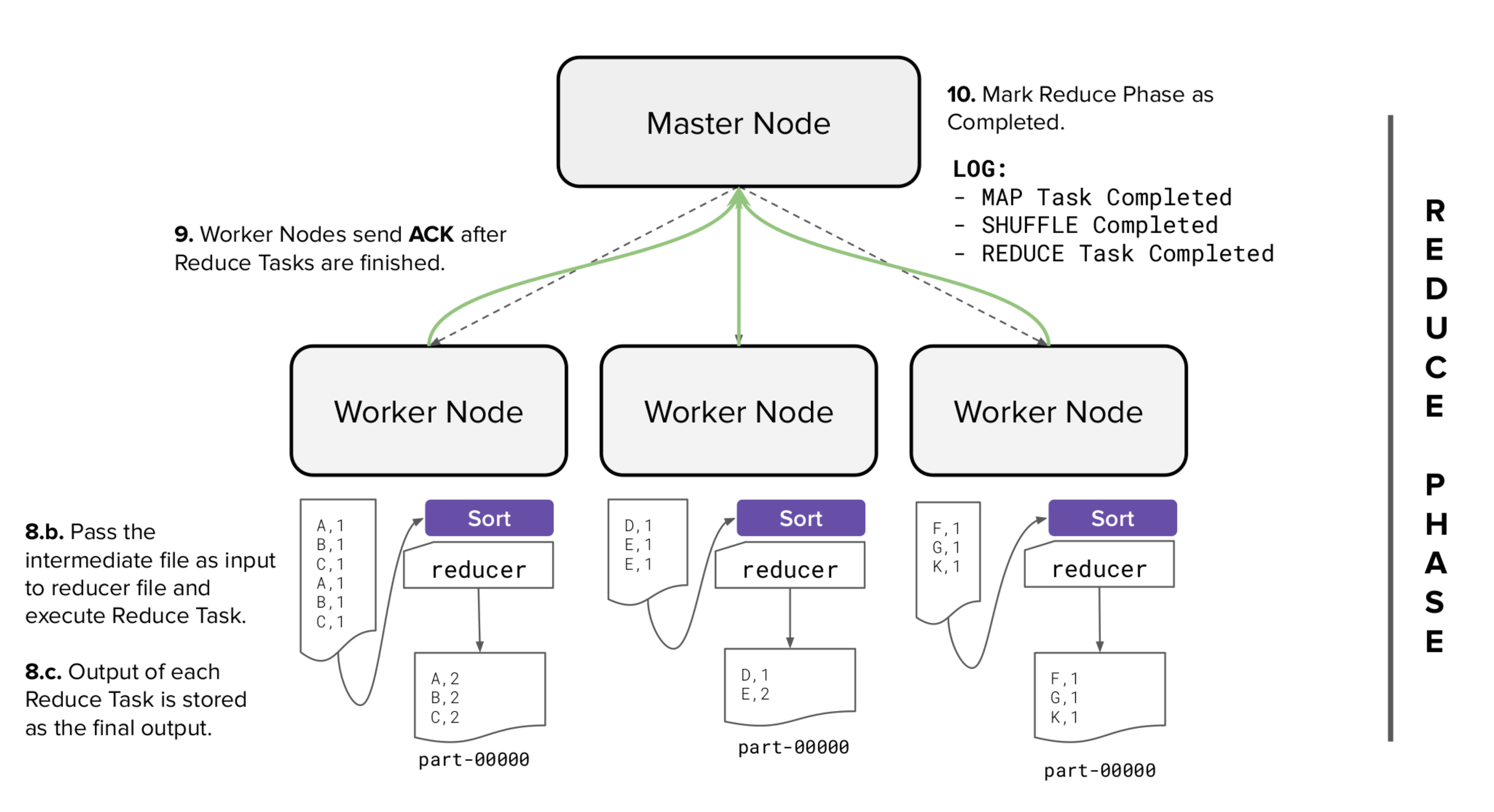
MAP-REDUCE operation workflow

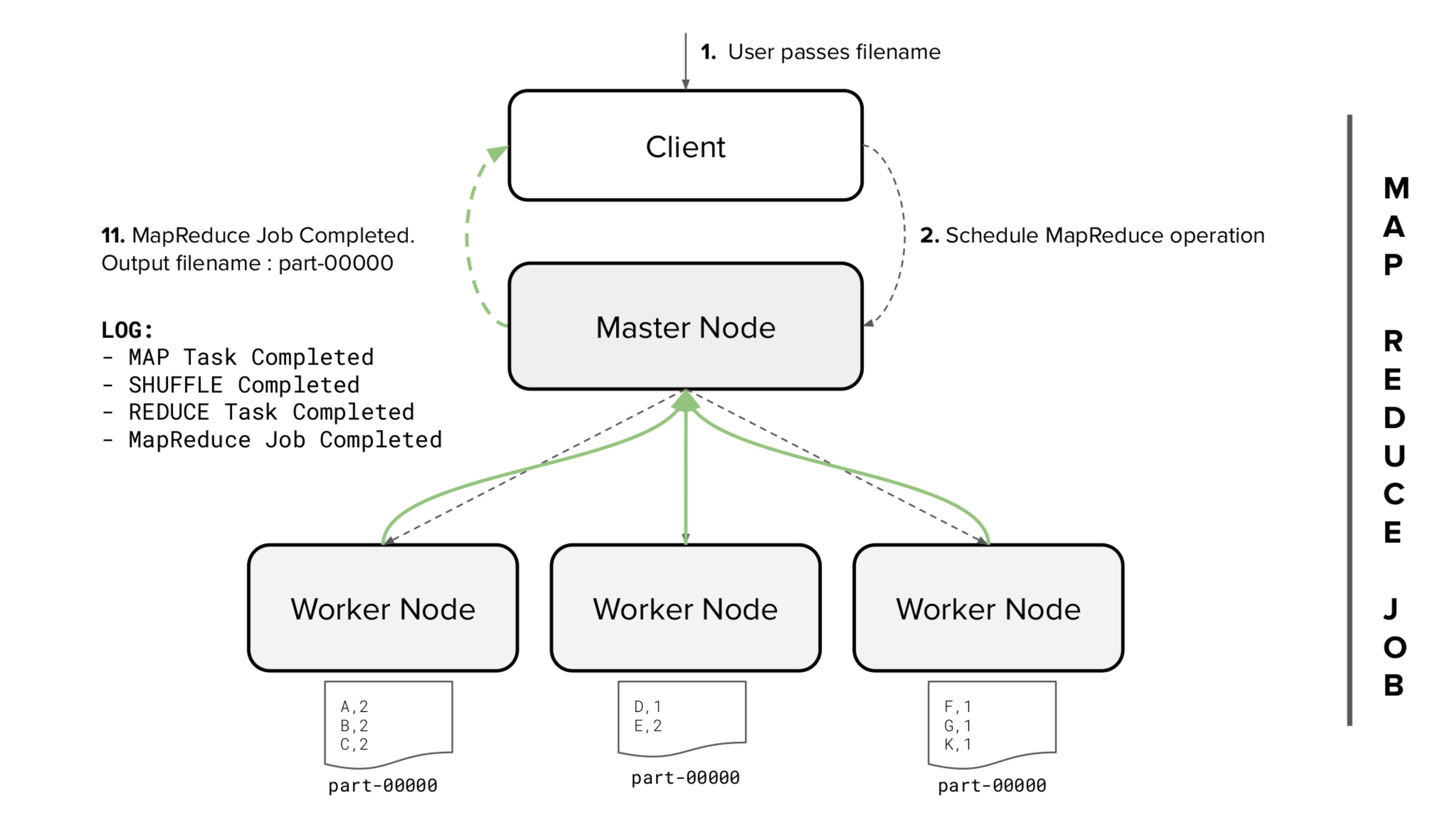












As shown in the image, the general workflow for a map-reduce operation is as follows:

- The user passes the filename, mapper file and the reducer file to the client.
- The client program contacts the master node to schedule the MAP-REDUCE operation.
- The rest of the workflow can be defined in 3 stages:
  - Map Stage
    - The master node checks the metadata to get a list of workers that have the partitions for the filename passed by the client.
    - To each node in the list, the master node passes the mapper file to execute the map task on the partition that the worker has.
    - Each worker executes the map task on the partition that it assigned to it and saves the output of the map task in an intermediate file.
    - The worker then sends an acknowledgement to the master that it has completed is task.
    - Once all the workers acknowledge that they have completed their map task, the Master node marks the Map stage as successful.
  - Shuffle Stage
    - After the completion of the map stage, the Master schedules the shuffle operation on all the workers.
    - In the Shuffle stage, the data in the intermediate file is distributed among the workers based on a hash function.
    - This hash function needs to be modular. The hash function should be swappable with any other valid hash function and your Shuffle operation must still produce the correct results.
    - Each worker stores the data that has been assigned by the hash function.
    - Once all the intermediate files are shuffled, the Master node marks the Shuffle stage as successful.
  - Reduce Stage
    - After the completion of the Shuffle stage, the Master passes the reducer file and schedules the reduce task on all the workers.
    - Each worker executes a Sort operation followed by the reduce task on the data that has been assigned to it in the Shuffle stage and saves the output of the operation to an output file.
    - The worker then sends an acknowledgement to the master saying that it has completed the reduce task.
    - Once all the workers acknowledge that they have completed their reduce task, the reduce operation and therefore the MAP-REDUCE operation is marked complete.
  - The Master then informs the client program that it has completed the MAP-REDUCE operation and passes it the output filename.
  - The client then performs a READ operation on the output filename to gather the final result for the MAP-REDUCE operation and displays it to the user.

Scope and Simplifying Assumptions

The simplifying assumption made in the workflows specified is that the input file is equally split among all the workers irrespective of the file size. That is, for an input file of size `S` with `W` workers in cluster, the size of each partition or block is `S / W`. It is also a given that the `Number of mappers = Number of reducers = W`.

To reduce the scope of the project, you are expected to only implement the basic features of Hadoop's HDFS and MapReduce Framework as listed in the previous sections.

The following functionalities are considered to be out of scope:

- Node failure
  - You are **NOT** expected to handle cases of Node failure. You can assume that all the nodes (Master and Worker) are functional at all points in time.
- Data Replication
  - You are **NOT** expected to store multiple replicas of the same partition
- Multiple MapReduce jobs at the same time
  - You are **NOT** expected to handle cases where the client submits multiple MapReduce jobs to the Master at the same time. Although, you will be evaluated on MapReduce jobs scheduled one after the other.

Tips and Guidelines

- To schedule the Map and Reduce tasks on the worker nodes, a module like `subprocess` in python, or an equivalent in the language of your choice can be used.
- The partition function should be modular. It should work for any key that is provided to it.
- Make sure that MapReduce jobs are independent of one another. Data of a run should not overwrite the data created in the previous runs.
- Edge cases within the scope of the project should be taken care of by proper error handling.

NOTE: Plagiarism of any form will **not be tolerated**. Using existing solutions on GitHub or co-operating with other teams is strictly not allowed.