High level design document of workers, masters, and their interaction and decision made when worker fails

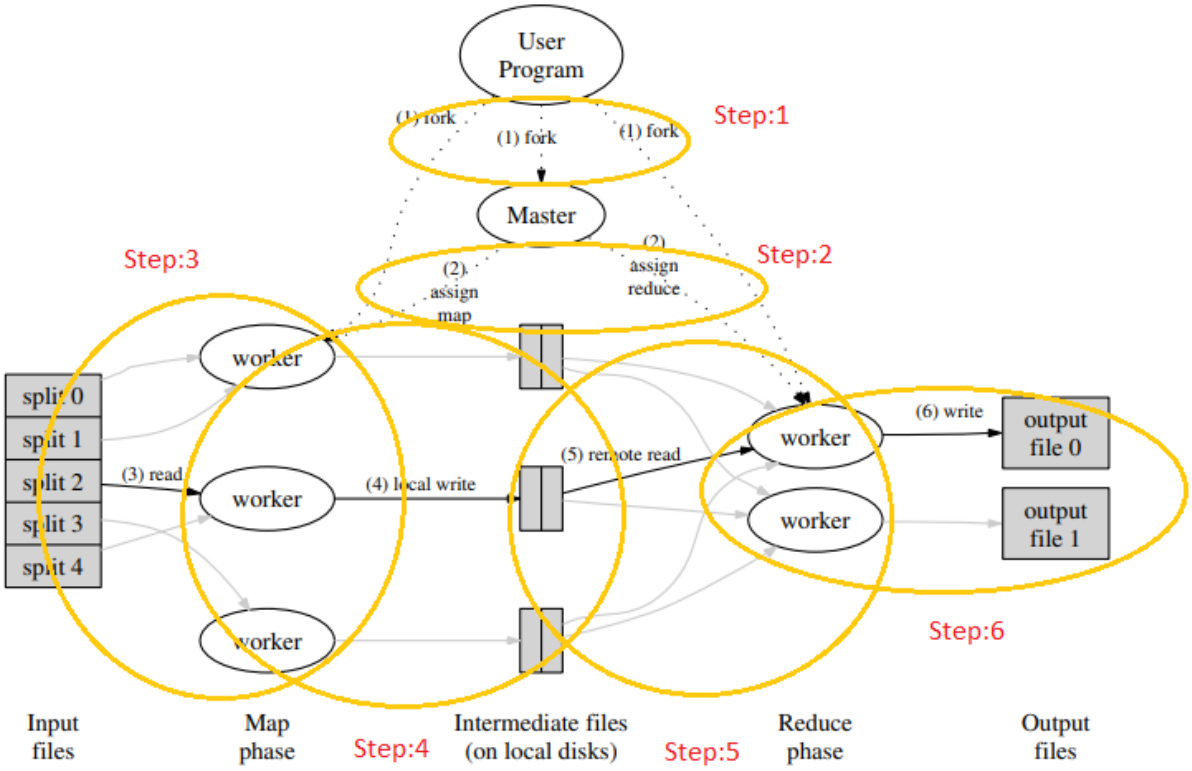

Figure 1: Execution overview

Design document of workers, masters, and their interaction

In the github Repository in folder name 2 the files are present with main folder names main and mapreduce

Main contains files wc.go , input.txt , test-wc.sh

Mapreduce contains files common.go , mapreduce.go , master.go, worker.go

Master distributes jobs to a set of workers using by RPC messages is present in our code in (common.go in the mapreduce package) and the code for called worker in (worker.go in mapreduce package)

The code in mapreduce.go contains mapreduce.Register RPC this function passes the worker information to mr.registerchannel

And information about Mapreduce job is in the Mapreduce struct that is defined in mapreduce.go

The master sends RPCs to the workers in parallel so that workers can work on jobs concurrently

The master may have to wait for a worker to finish their corresponding works

The channels are useful in synchronize the threads

First the master is called with number of maps and no of reducers then the file file is divided into Fixed sizes and assigned to map workers by balancing load across the workers and the Workers concurrently execute same map functions parallely and produce files passed to reduce Workers and output files are generated

if the master's RPC to the worker fails, the master should re-assign the job given to the failed worker to another worker who is idle in idle workers channel

And if the worker not failed but cannot able to communicate with master with respective network delay in the message passing and if the same work is assigned to new worker then two workers doing the same task.however there is no problem because jobs are idempotent, it doesn't matter if the same job is computed twice - both times it will generate the same output

We don't have to handle failures of the master; we will assume that it won't fail. And also making Master fault tolerance is more difficult because the master keeps persistent state that must be replicated in order to make master fault tolerant