Project 1 - Documentation

Alok Kumar
Ritvika Pillai
Siddharth Raman

11/3/2021

1. About the project

The main idea behind the project is to implement a standalone version of Google's MapReduce API running on a single server with multiprocessing and fault tolerance.MapReduce is basically a programming model which was developed to handle humongous data and process them in an efficient way by using a distributed system. A pair of key/value pairs is given as an input to the model which in turn produces output key/value pairs. The operation is divided into 2 functions i.e map and reduce. The Map functionality is written by the user which takes input pairs and produces intermediate pairs. These are then grouped further and passed on to the reduce function. The Reduce functionality is also written by the user and takes up the set of intermediate values for a key as input. It then merges them to form a smaller value.

2. System Design

2.1 Tech Stack

- Python
- Multiprocessing Module
- OS Module
- Process Pool Executor
- Bash Script

2.2 Architecture

The complete architecture of the system is shown in the diagram below.

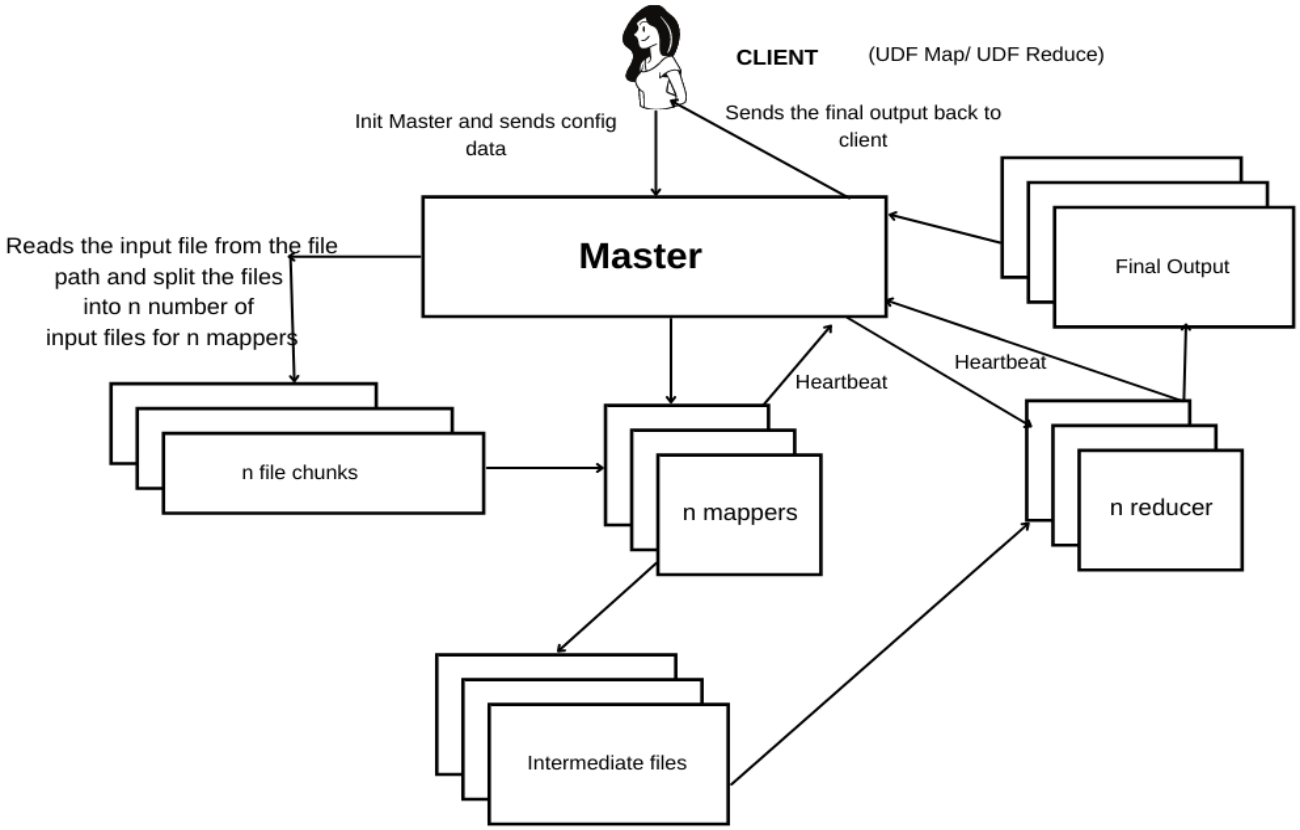

Fig 2.1 : Overview of the design of the system

Explanation of the architecture:

1. The client invokes master and passes both user-defined map and user-defined reduce function along with all the other configs.
2. Master reads the file from the input path and then splits it into n number of data chunks file where n is the number of mappers defined by the user

3. Then master spawns n number of mapper processes and one mapper is assigned to for each chunk of data.
4. Once the mapper finished their jobs they write their intermediate output in such a way that one key is assigned to only one reducer. This is achieved by using hashing the key and taking the module by the number of reducers. Mappers after finishing the map phase write their data on an intermediate file.
5. Once all the mappers finish their job. Master respawns the reducers and the reducers read the intermediate file in form of key-value pair and generate the final output.

Our map-reduce system is fault-tolerant which means that if a worker dies master will respawn a new worker and transfer the workload from the old worker to the new one. As our map-reduce job is running on a very small scale so the chance of a process dying on its own is almost negligible so to demonstrate fault tolerance we have passed a kill flag in one of our test cases while initializing master. If the kill flag is set then the master will kill the worker with the corresponding flag id and then restart a new process.

Test Cases

1. **calculate_percentage**
   In the mapper phase, each line of the dataset is split into names and scores. The name is mapped to each value of score and sent to the reducer as a key/value pair. The reducer takes the values(i.e score) corresponding to a particular key (i.e name) and calculates the percentage score. In the traditional method by reading the complete file and calculating the percentage. The output of the traditional computation is compared with the output of the reducer.

2. **test_word_count**
   In the mapper phase, the data file is split and each word is assigned a value of 1. These key/value pairs are sent to the reducer. The reducer takes the value(i.e 1) mapper to each key(i.e word) and adds it to the result to get the final count. The output of the reducer is then compared to the traditional computation of word count.

3. **test_word_count_with_fault_tolerance**

This is test script is somewhat similar to the word count test. Although here we are passing a kill index while initializing the master to demonstrate the fault tolerance. The process corresponding to that particular index is killed. The master will respawn a new process and the workload of the old process is assigned to this new process. Also, all the computations that might have already been performed by the old process will be discarded.

4. **test_grouping**
   In the mapper phase, the data is split into age as the key and the name of the person as value. The key-value pair is then sent to the reducer. The values(i.e names) grouped together based on the same key (i.e age) are sorted and given out as the reducer output. This output is compared with the output of the traditional computation.