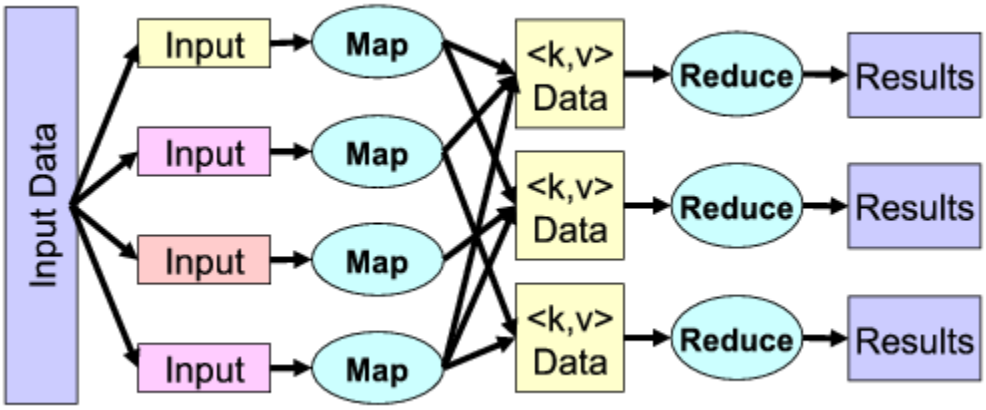
Brief explanation of Map-reduce

- MapReduce is programming model that uses distributed, parallel processing on large data sets. The MapReduce framework mainly consists of two major operation mapping and reducing.
- Mapper performs sorting and filtering and reducer performs summary operation on the input from the mapper.
- The Mapper Reducer works on the master slave based principle. The master distributes the job among various slaves which can act as mapper or reducer.



- The above diagram shows the working of a typical MapReduce, large data is divided into smaller parts for processing by the mapper and reducer.
- The mapper returns a key value pair which is sorted and given to the reducer for further processing and then the result is written to the output.

Pseudo code:

MapReduce Job #1

Mapper #1 // Compute average per line

Input<FileID_LineNumber , Line>
Int count=0; // count is used for total number of integers in a line .Initially set to 0.

Int sum=0; // result of adding all the integers is stored in sum

float avg; // Stores the average

String Tokens[]= line.tokenzie("\t"); // Tokenize the line to get each integer

For(token:Tokens) //Repeat the loop until tokens exist

sum= sum + GetInteger(token); // GetInteger converts string to int

count++;

end for;

avg = sum / count; //get the avg ;

Emit(FileID,avg); // output of Mapper #1

Reducer #1 //Calculate average per file

Input<FileID , Float [ ]values>

Int count=0; // count is used for total number lines in a file.

float sum=0; // result of adding all the avgs is stored in sum

float avg; // Stores the average of the file

For(Float val:values) //Repeat the loop until values exist

sum= sum +val; // GetInteger converts string to int

count++;

end for;

avg = sum / count; //get the avg ;

Emit(FileID,avg); // output of Mapper #1

MapReduce Job #2

Mapper #2 // Emit output as it is

Input<FileID,avg >

Emit(1,avg);

Reducer #2 //Calculate average of all file

Input<1, Integer [ ]values>

Int count=0; // count is used for total number of file.

float sum=0; // result of adding all the avgs is stored in sum

float avg; // Stores the average of the file

For(Float val:values) //Repeat the loop until values exist

sum= sum +val; // GetInteger converts string to int

count++;

end for;

avg = sum / count; //get the avg ;

Emit(1,avg); // output of Mapper #1

Explanation of each function used

In the above example we have to calculate the average of all numbers present in multiple files. Thus, we will require 2 levels of Mapper Reducer for calculating the average.

- Level 1
  The level 1 mapper reducer calculates the average in a single file.
  - Mapper #1:

    Input: <FileID_lineNumber, line>.

    Processing done: The mapper tokenizes the line to get each integer in the line. The count of the number of Integers in the line is calculated. The average is calculated for each line in the file by the mappers running in parallel. The output for each mapper is the fileId and average calculated.

    Output: <FileID, average>

  - Reducer #1

    Input: <FileID , value[]>

    Processing done: The reducer collects all the averages calculated by the mapper the same fileid and calculates the average of it. Thus reducers emits output with average for the all integers in the file.

    Output: <FileID, average>

- Level 2
  The level 2 mapper reducer calculates the average for all files after it gets average for each file.

- Mapper #2

  Input < FileID, average>

  Processing done: The mapper just forwards the output to the reducer by combing .

  Output<1, average>

- Reducer #2

  Input: < 1,values[]>

  Processing done: The reducers collects the averages calculated for each file and then computes the average for all the files. The output is the average of all integers in the file.

  Output: <1, average>

Sample Result using Pseudo code

If we assume 3 files with 4 integers

File 1: 1 2

3 4

File 2: 5 6

7 8

File 3: 9 10

11 12

At level 1

Mapper 01 for file 1 output: <1, 1.5>

Mapper 02 for file 1 output: <1, 3.5>

Reducer for file 1 Output : <1, 2.5>

Mapper 01 for file 2 output: <2, 5.5>

Mapper 02 for file 2 output: <2, 7.5>

Reducer for file 2 Output : <2, 6.5>

Mapper 01 for file 3 output: <3, 9.5 >

Mapper 02 for file 3 output: <3, 11.5 >

Reducer for file 3 Output : <3, 10.5>

At Level 2

Mapper for file 1 output: <1, 2.5>

Mapper for file 2 output: <1, 6.5>

Mapper for file 3 output: <1, 10.5>

Reducer Input: <1, [2.5, 6.5, 10.5]>

Reducer for final Output : <1, 6.5>