Project 1

**B649: CLOUD COMPUTING**

# Sohil Jain (sohjain@iu.edu)

# Ashutosh Bhargave (ashubhar@iu.edu)

We have implemented the concept of a Combiner to optimize the statistical calculation in this program. We have also implemented the NlineImputFormat implementation to spawn multiple mappers and make our execution parallel(Only the Average function). Details about the program are described below

## Transformation of Data during Computations

Below we are describing the datatypes of <key, value> pairs as inputs and outputs to the Mapper, Combiner and Reducer.

1. Mapper:

a. Input - <key, value>

i. Key - LongWritable

ii. Value - Text

b. Output -

i. Key - Text

ii. Value – Text

2. Combiner:

a. Input: Takes input from output of mapper tasks.

i. Key: Text

ii. Value: Text

b. Output:

i. Key: Text

ii. Value: Text

3. Reducer:

a. Input: Takes input from output of combiner tasks.

i. Key: Text

ii. Value: Text

b. Output:

i. Key: Text

ii. Value: DoubleWritable

## The Data Structure used to transfer between Map and Reduce phases:

Following are the Data Structures I have used in the Mapper, Combiner and Reducer Phases -

**Mapper<LongWritable, Text, Text, Text>**

My mapper input output is as described above. I take input as <LongWritable, Text>. and the output are both Text. I send the value read from the text file, as both key and value in my mapper output.

context.write(new Text(value.toString()), new Text(value.toString()));

- Input 1000 records, Output 1000 records. The key and value both have same Text as the value.

**Combiner<Text, Text, Text, Text>**

Next, I’ll use a combiner to aggregate on the basis of these values. So, what will happen is if I get a key with value 0.34, I’ll combine all the values corresponding to this key in the combiner and then send the values to the reducer. From my combiner, I’ll send same key, and aggregated values to the reducer.

result = new Text(sum + " " + sumSq + " " + min + " " + max + " " + count);

context.write(new Text("k"), result);

– Input 1000 records, Output 99 records. There are total 99 unique values in the data so there should be 99 different keys the combiner will receive and aggregate their result to the reducer.

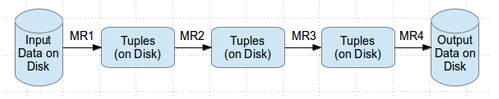
**Reducer<Text,Text,Text,DoubleWritable>**

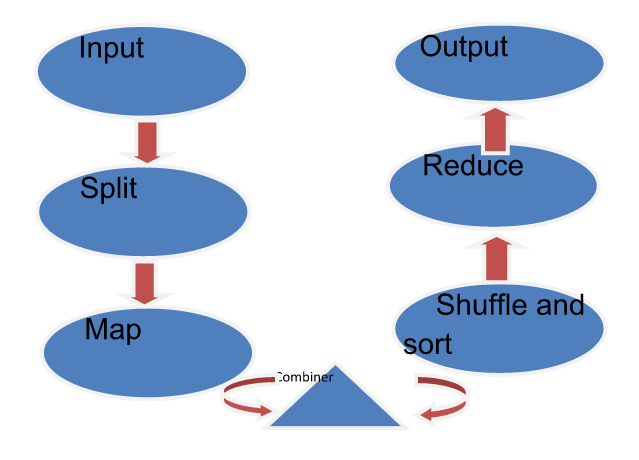
Reducer will take all these values for the same key, and perform aggregations.

– Input 99 records, Output 1 record. The reducer will read the 99 unique key records sent by the user. They all have the same key set as “k”, so the reducer will aggregate all the values for this key (which means all the values) and give only one output. (I have attached the snapshot of my output in the end.)

## How the data flow happens through disk and memory during the computation

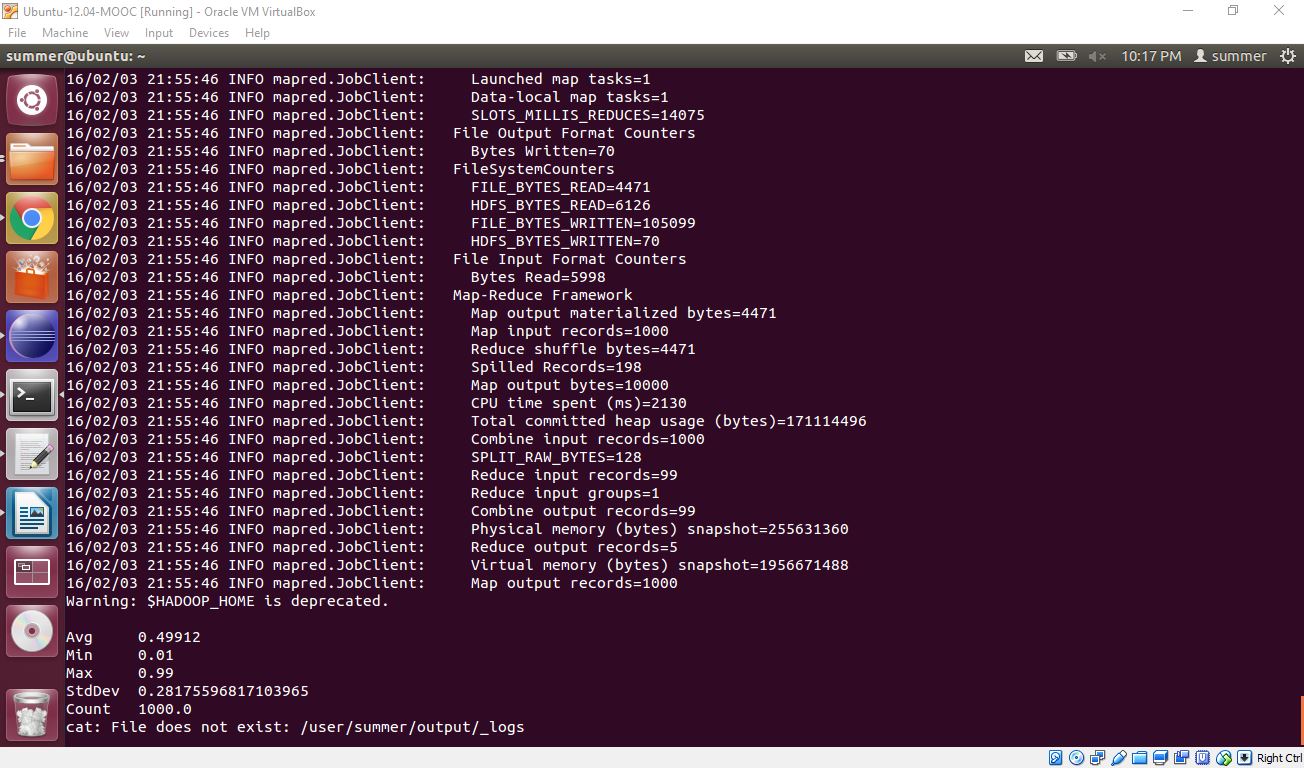
HDFS is a Distributed File System, and Map-Reduce works in a distributed way where each node has its own memory and disk. The data is read from the disk, and processed in memory in every phase of Map-Reduce, then it is spilled to the disk. This makes it scalable, but also adds an overhead of I/O which is not good in use cases like Machine learning where iterative programs are executed. Spark is a good implementation of Iterative Map-Reduce in which in-memory computing can happen.





* **Input:**This is the input data / file to be processed.
* **Split:**Hadoop splits the incoming data into smaller pieces called "splits".
* **Map:**In this step, MapReduce processes each split according to the logic defined in map() function. Each mapper works on each split at a time. Each mapper is treated as a task and multiple tasks are executed across different TaskTrackers and coordinated by the JobTracker.
* **Combine:**This is an optional step and is used to improve the performance by reducing the amount of data transferred across the network. Combiner is the same as the reduce step and is used for aggregating the output of the map() function before it is passed to the subsequent steps.
* **Shuffle & Sort:**In this step, outputs from all the mappers is shuffled, sorted to put them in order, and grouped before sending them to the next step.
* **Reduce:**This step is used to aggregate the outputs of mappers using the reduce() function. Output of reducer is sent to the next and final step. Each reducer is treated as a task and multiple tasks are executed across different TaskTrackers and coordinated by the JobTracker.
* **Output:**Finally the output of reduce step is written to a file in HDFS.

## Output



Avg 0.49912  
Min 0.01  
Max 0.99  
StdDev 0.28175