K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: BDA_2 Roll No.: 1211061

Experiment / assignment / tutorial No.__2___

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementation of simple programs on MapReduce (Word count, Maximum temperature etc.)

AIM : To Implementation of simple programs on MapReduce (Word count, Maximum temperature etc.)

Expected Outcome of Experiment :

CO :

CO2: Understand the fundamental enabling techniques like Hadoop, MapReduce and NO SQL in achieving Big data analytics

---

Books/ Journals/ Websites referred :

1. Anand Rajaraman and Jeff Ullman “Mining of Massive Datasets”, Cambridge University Press,
2. Alex Holmes “Hadoop in Practice”, Manning Press, Dreamtech Press.
3. Big data analytics by Radha Shankarmani, M. Vijayalakshmi. Wiley publication

---

Pre Lab/ Prior Concepts:

Map Reduce:

- Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
- A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.
- Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.
- The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.
- Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*. The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the JobTrackerwhich then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic.

<u>Map and Reduce Function for Wordcount program:</u>

Mapper:

```
        public void map(LongWritable ikey, Text ivalue, Context context)
                        throws IOException, InterruptedException {
                String line=ivalue.toString();

                String[] tokens=StringUtils.split(line,' ');
                System.out.println(Arrays.toString(tokens));
                int size=tokens.length,c=1;

                for(int i=0; i<size; i++)
                {
                        context.write(new Text(tokens[i]), new IntWritable(c));
        }

    }
```

Reducer:

```
    public void reduce(Text _key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        // process values

        Iterator<IntWritable> iterator=values.iterator();

        int m=0;
        while(iterator.hasNext())
        {
            m=m+iterator.next().get();
        }

        //System.out.println(m); context.write(_key,
        new IntWritable(m));
    }
```

Map and Reduce Function for Max temperature program:

Mapper:

```
    public void map(LongWritable ikey, Text ivalue, Context context)
            throws IOException, InterruptedException
    {
        String line=ivalue.toString();

        String[] tokens=StringUtils.split(line, '|');
        if(tokens.length ==5)

        {
            int year=Integer.parseInt(tokens[1]);
            float temp=Float.parseFloat(tokens[4]);
            //write the output to the mapper
              context.write(new IntWritable(year), new FloatWritable(temp));

        }
    }
```

Reducer:

```
public void reduce(IntWritable _key, Iterable<FloatWritable> values, Context
                context) throws IOException, InterruptedException {
        // process values

        Iterator<FloatWritable>
        iterator=values.iterator(); float
        maxtemp=0;

        if(iterator.hasNext())
        {
                maxtemp=iterator.next().get();

        }
        while(iterator.hasNext())
        {
        float temp=iterator.next().get(); if(temp> maxtemp)
                {
                        maxtemp=temp;
                }
        }
        context.write(_key, new FloatWritable(maxtemp));
}
```

Conclusion:

Thus we understood the concept of map reduce and Implemented simple programs on MapReduce Word count, and map reduce Maximum temperature

Post Lab Questions:

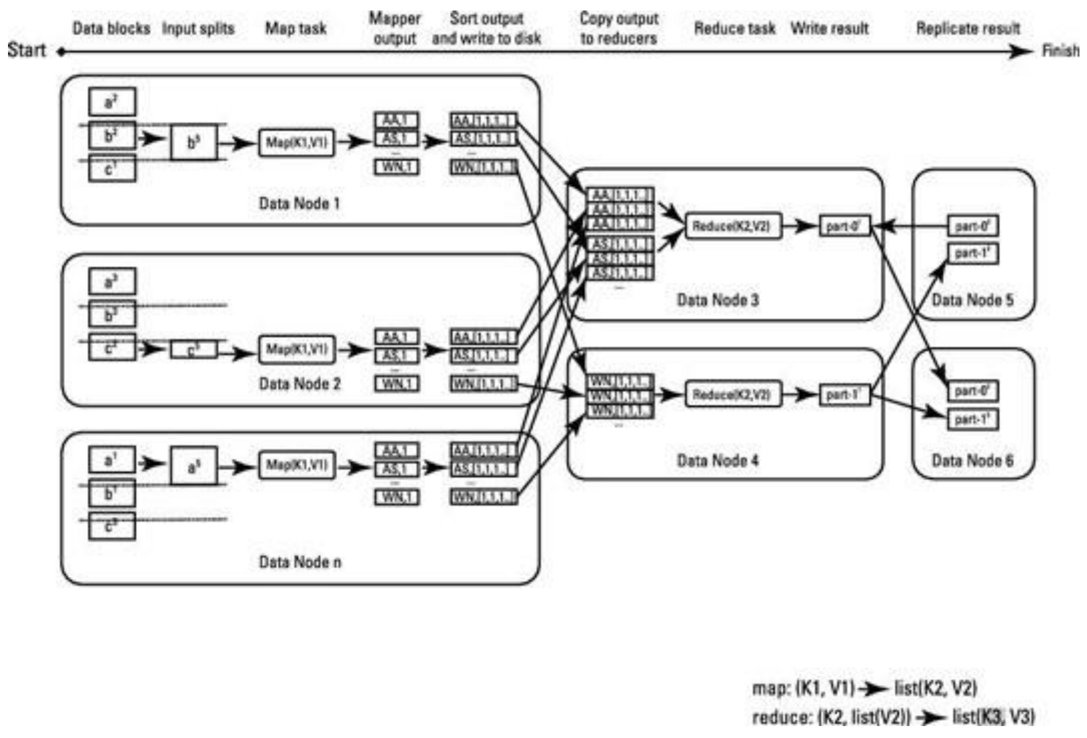
1. **What is the use of Shuffling in Map Reduce.**

A:

- After the Map phase and before the beginning of the Reduce phase is a handoff process, known as *shuffle and sort*. Here, data from the mapper tasks is prepared and moved to the nodes where the reducer tasks will be run. When the mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk.
- You can see this concept in the following figure, which shows the MapReduce data processing flow and its interaction with the physical components of the Hadoop cluster. (One quick note: Data in memory is

represented by white squares, and data stored to disk is represented by gray squares.)

- To speed up the overall MapReduce process, data is immediately moved to the reducer tasks' nodes, to avoid a flood of network activity when the final mapper task finishes its work. This transfer happens while the mapper task is running, as the outputs for each record — remember — are stored in the memory of a waiting reducer task.



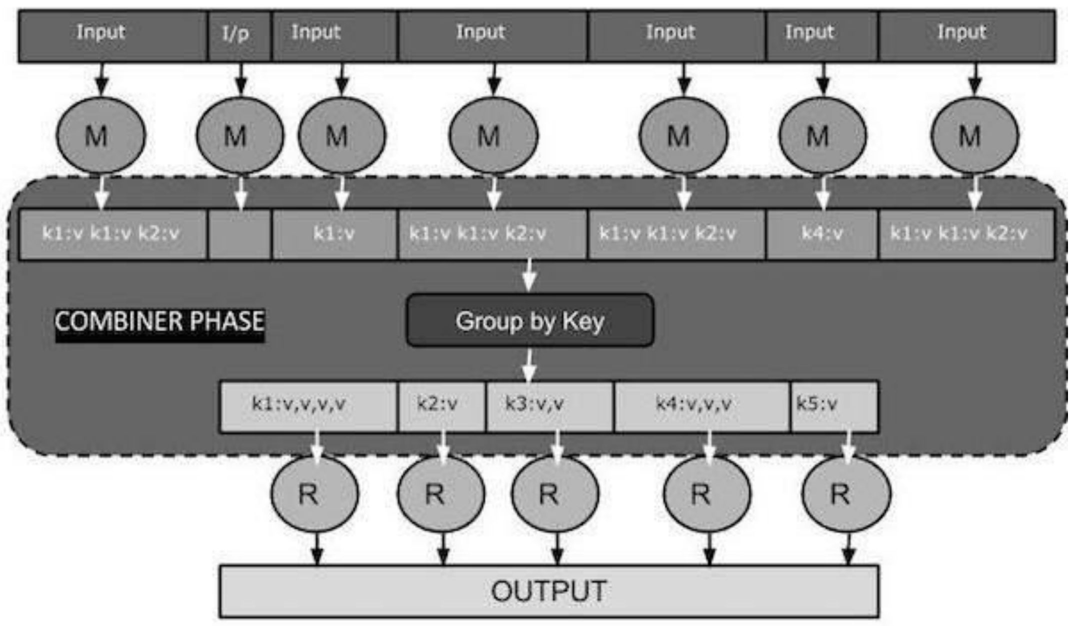
2. What are combiners?. When can we use combiners in Map Reduce job?
A:

- A Combiner, also known as a **semi-reducer,** is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.

- The main function of a Combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.

- The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

- The following MapReduce task diagram shows the COMBINER PHASE.



<u>Comibner works on following way :</u>

- A combiner does not have a predefined interface and it must implement the Reducer interface’s reduce() method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.
- A combiner can produce summary information from a large dataset because it replaces the original Map output.