**Lecture 4**

Steps for Map Reduce in Hadoop

1. Pick an appropriate InputFormat( If none is set then default is Text InputFormat)
2. FileInputFormat<**LongWritable, Text**>
3. Define a Mapper class
4. Mapper<**KEYIN, VALUEIN**, **KEYOUT, VALUEOUT**> The fileinput must match mapper input
5. Define a Reducer Class
6. Reducer<**KEYIN, VALUEIN,** **KEYOUT,VALUEOUT**> Input same as mapper output
7. Pick an appropriate output format
8. OutputFormat<**KEY,VALUE**>

Reducer only start until all mapper is finished because of sorting and shuffling

Output of map is stored locally on datanode

Moved to HDFS once finished.

Then brought to reducer

Ideally- no of reducer= no of keys

**Partition**

**K1→ 1 million values**

**K2→ 1 billion values**

**Hash Keys % No of reducers**

Partitioner class decide which keys will be sent to which reducer.

Public class MyPartitioner extends Partitioner{

@Override

int getPartition(Key key, Value value, int numPartitions){

// return hash(key)%numPartitions;

if(key==K1)return 0;

else return (int)value/1 000 000 + 1;

}

}

Divide them equally or add more workers

Adding more workers is always better.

**OPTIMIZATION**

1. **job.setPartitionerClass(myCustomPartitioner);**
2. **job.setCombinerClass(myCombiner);**

**If the operation is both COMMUTATIVE and ASSOSIATIVE then same reducer could be used as a COMBINER**

**Q—count number of lines of a text file using MR**

TextInputFormat

**public class** LineCountMapper **extends** Mapper {  
   
 @override  
 **public void** map(Key key, Value value, ){  
 context.write(NullWritable.get(), **new** IntWritable(1));  
 }  
}

Methode override from Mapper class

Setup—start

Map—to map key and pair

Cleanup—to close

**getmerge()**

To merge files in a directory

Hadoop fs -getmerge -nl /src /opt/output/output.txt

**putmerge()**

cat NYSE\_daily\_prices>\*.csv > nysemergedfile.txt