**MapReduce**

**MapReduce** is the processing layer of **Hadoop**. MapReduce programming model is designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks.

**InputFormat**

- Default TextInputFormat

- split the input into logical InputSplits based on total size of the input files.

- filesystem blocksize is the upperbound and lower bound is set with mapreduce.input.fileinputformat.split.minsize

**InputSplit**

- Data to be processed by individual mapper

- presents byte-oriented view of the input

- FileSplit is the default InputSplit

**RecordReader**

- presents record oriented view of the input

- Reads <key, value> pairs from InputSplit

**Mapper**

- Equal to total size of inputs. Total number of blocks of input files

- 10-100 maps per node

- Configuration.set(MRJobConfig.NUM\_MAPS, int)

**Combiner**

- Also known as mini-reducer

- reduces the time taken for data transfer between mapper and reducer

- improves performance of reducer as the amount of data transferred to reducer is decreased.

**Partitioner**

- Partitioning of the keys of the intermediate map output is controlled by the Partitioner.

- hash function is used to derive the partition with the key

- hash partitioner is the default partitioner

- Partitioner will divide the data according to the number of reducers which is set by *JobConf.setNumReduceTasks()* method

**Reducer**

- Job.setNumReduceTasks(int)

- 0.95 or 1.75 \*(<no of nodes> \* <no of maximum containers per node>)

**OutputFormat**

- validate output-specification of the job

- provide recordReader implementation used to write the output files of the job.

- TextOutputFormat is default

**RecordWriter**

- writes output <key, value> pair to output file

**Word count example**

import java.io.IOException;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

public class WordCount {

public static class Map extends MapReduceBase implements

Mapper<LongWritable, Text, Text, IntWritable> {

@Override

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter)

throws IOException {

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

value.set(tokenizer.nextToken());

output.collect(value, new IntWritable(1));

}

}

}

public static class Reduce extends MapReduceBase implements

Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterator<IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter reporter)

throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

output.collect(key, new IntWritable(sum));

}

}

public static void main(String[] args) throws Exception {

JobConf conf = new JobConf(WordCount.class);

conf.setJobName("wordcount");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);

conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}