MR-(Mapreduce Programming Language)

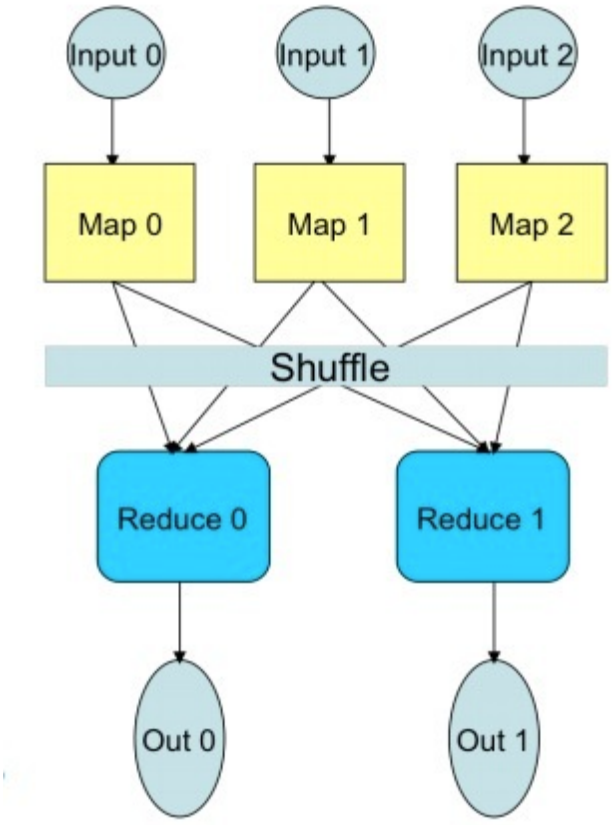
Siyang Dai sd2694
Zhi Zhang zz2219
Shuai Yuan sy2420
Zeyang Yu zy2156
Jinxiong Tan jt2649

Objective of MR

MapReduce is a software framework introduced by Google, aiming at processing distributed computation with huge datasets on a large number of computing nodes. However, this framework includes many duplicable and routine code segments in terms of configuration and exception handling, which is unnecessary burden and distraction for software developer.

Our proposed MR Programming Language seeks to provide programmers with a kind of concise and powerful language that simplifies the develop process. Programmers only have to focus on the *Map* function and *Reduce* function and specify configuration value, our compiler will finish the rest of work.

The basic framework of Mapreduce:



Basic Types

byte
boolean
int
long
float
double
text

Statements

Configuration Statement (start with a '#')
Definition
Assignment
Emit Statement
If
While
For

Word Count Example:
In MR:

```
#JobName = “word count”
#Input = $[1]
#Output = $[2]
#Mappers = $[3]
#Reducers = $[4]
#Combiners = $[5]

// variable “key” and “value” are default in Map and Reduce functions, just like the this pointer in OO.
def mapfunction( <long, text>, <text, int> ): Map
        words = value.split()
        for word in words:
                emit(word, 1)
def reducefunction( <text, int>, <text, int> ): Reduce, Combine
        word = key
        count = value.size()
        emit(word, count)
```

But in Java:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
/**
 * This is an example Hadoop Map/Reduce application.
 * It reads the text input files, breaks each line into words
 * and counts them. The output is a locally sorted list of words and the
 * count of how often they occurred.
 *
 * To run: bin/hadoop jar build/hadoop-examples.jar wordcount
 *            [-m <i>maps</i>] [-r <i>reduces</i>] <i>in-dir</i> <i>out-dir</i>
 */
public class WordCount extends Configured implements Tool {

  /**
   * Counts the words in each line.
   * For each line of input, break the line into words and emit them as
   * (<b>word</b>, <b>1</b>).
   */
  public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }

  /**
   * A reducer class that just emits the sum of the input values.
   */
  public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    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));
  }
}

static int printUsage() {
  System.out.println("wordcount [-m <maps>] [-r <reduces>] <input> <output>");
  ToolRunner.printGenericCommandUsage(System.out);
  return -1;
}

/**
 * The main driver for word count map/reduce program.
 * Invoke this method to submit the map/reduce job.
 * @throws IOException When there is communication problems with the
 *                     job tracker.
 */
public int run(String[] args) throws Exception {
  JobConf conf = new JobConf(getConf(), WordCount.class);
  conf.setJobName("wordcount");

  // the keys are words (strings)
  conf.setOutputKeyClass(Text.class);
  // the values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(MapClass.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  List<String> other_args = new ArrayList<String>();
  for(int i=0; i < args.length; ++i) {
    try {
      if ("-m".equals(args[i])) {
        conf.setNumMapTasks(Integer.parseInt(args[++i]));
      } else if ("-r".equals(args[i])) {
        conf.setNumReduceTasks(Integer.parseInt(args[++i]));
      } else {
        other_args.add(args[i]);
      }
    } catch (NumberFormatException except) {
```

```
        System.out.println("ERROR: Integer expected instead of " + args[i]);
        return printUsage();
      } catch (ArrayIndexOutOfBoundsException except) {
        System.out.println("ERROR: Required parameter missing from " +
                           args[i-1]);
        return printUsage();
      }
    }
    // Make sure there are exactly 2 parameters left.
    if (other_args.size() != 2) {
      System.out.println("ERROR: Wrong number of parameters: " +
                         other_args.size() + " instead of 2.");
      return printUsage();
    }
    FileInputFormat.setInputPaths(conf, other_args.get(0));
    FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));

    JobClient.runJob(conf);
    return 0;
  }


  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new WordCount(), args);
    System.exit(res);
  }

}
```