MapReduce Code
Word Count example

Designing the Mapper

- Define a mapper class that extends hadoop Mapper
  - imported from Hadoop MapReduce library org.apache.hadoop.mapreduce.Mapper;
  - format:
    - Mapper<inKey, inValue, outKey, outValue>
  - has a method called map

```
public static class TokenizerMapper
   extends Mapper<LongWritable, Text, Text, IntWritable>{
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  @Override
  public void map(LongWritable key, Text value, Context context
    ) throws IOException, InterruptedException {
   StringTokenizer itr = new StringTokenizer(value.toString());
   while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
   }
  }
}
```

Designing the Mapper

- Input key and input value types depends on the used input format
- We will use TextInputFormat to generate inputs for the map
    - key is the line offset (type is LongWritable) —> mapper input key = LongWritable
    - value is the content of the line (type is Text) —> mapper input value = Text

```
public static class TokenizerMapper
   extends Mapper<LongWritable, Text, Text, IntWritable>{
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();


  @Override
  public void map(LongWritable key, Text value, Context context
     ) throws IOException, InterruptedException {
   StringTokenizer itr = new StringTokenizer(value.toString());
   while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
   }
  }
 }
```

Designing the Mapper

- Output key is Text which is initialized with value equal to 1
- Output value is IntWritable which contains one of the words in the line
- context is used to write the intermediate key-value pair

```
public static class TokenizerMapper
     extends Mapper<LongWritable, Text, Text, IntWritable>{
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   @Override
   public void map(LongWritable key, Text value, Context context
      ) throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
     }
   }
 }
```

Designing the Reducer

- Define the reducer class that extends hadoop Reducer
  - must specify the type of the input/output key-value pairs
  - override the reduce function
  - Mapper output key = Text —> Reducer input key = Text
  - Mapper output value = IntWritable —> Reducer input value = IntWritable

```
public static class IntSumReducer
   extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();
  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
     Context context) throws IOException, InterruptedException {
   int sum = 0;
   for (IntWritable val : values) {
     sum += val.get();
   }
   result.set(sum);
   context.write(key, result);
  }
 }
```

Designing the Reducer

- reduce method iterates over all values associated with one key in a for loop

- sum values to generate the word count

```
public static class IntSumReducer
   extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();
  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
    Context context) throws IOException, InterruptedException {
   int sum = 0;
   for (IntWritable val : values) {
    sum += val.get();
   }
   result.set(sum);
   context.write(key, result);
  }
 }
```

Designing the Reducer

- output (key, value)
  - key is equal to the input key (word) of type Text
  - value is the result of summing all values of type IntWritable
- types of the output keys and values must match the types configured for the job output
- use the context to write the final output

```
public static class IntSumReducer
   extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();
  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
    Context context) throws IOException, InterruptedException {
   int sum = 0;
   for (IntWritable val : values) {
    sum += val.get();
   }
   result.set(sum);
   context.write(key, result);
  }
}
```

The Job Configuration/ job driver

- The configuration is important because it indicates how the job is submitted

- make an instance of the job, give it a name

- tell hadoop which jar to run, the one that contains our main class

```
public class WordCount {

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setInputFormatClass(TextInputFormat.class)
   job.setOutputFormatClass(TextOutputFormat.class)
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
//Mapper
public static class TokenizerMapper
     extends Mapper<LongWritable, Text, Text, IntWritable>{
…..
}
//Reducer
public static class IntSumReducer
     extends Reducer<Text, IntWritable, Text, IntWritable> {
…..
}
}
```

The Job Configuration

- configure the job mapper, combiner (optional), reducer classes
- combiner can be the reducer

```
public class WordCount {

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setInputFormatClass(TextInputFormat.class)
   job.setOutputFormatClass(TextOutputFormat.class)
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
//Mapper
public static class TokenizerMapper
     extends Mapper<LongWritable, Text, Text, IntWritable>{
…..
}
//Reducer
public static class IntSumReducer
     extends Reducer<Text, IntWritable, Text, IntWritable> {
…..
}
}
}
```

The Job Configuration

- Set the type of the final output key-value pairs

```
public class WordCount {

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setInputFormatClass(TextInputFormat.class)
   job.setOutputFormatClass(TextOutputFormat.class)
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
//Mapper
public static class TokenizerMapper
    extends Mapper<LongWritable, Text, Text, IntWritable>{
…..
}
//Reducer
public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
…..
}
}
}
```

The Job Configuration

- Input / output configuration
  - input format class
    - here we use TextInputFormat class
  - output format class
    - TextOutputFormat class
  - path pointing the the input data
    - read from the command line
  - path to where final output should be written
    - read from the command line

```
public class WordCount {

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setInputFormatClass(TextInputFormat.class)
   job.setOutputFormatClass(TextOutputFormat.class)
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
//Mapper
public static class TokenizerMapper
     extends Mapper<LongWritable, Text, Text, IntWritable>{
…..
}
//Reducer
public static class IntSumReducer
     extends Reducer<Text, IntWritable, Text, IntWritable> {
…..
}
}
```

The Job Configuration

- Submit the job and wait until it finishes
- To lunch the job
  - create a jar that contains the code

```
> hadoop jar “jar-file-name” “main class” “path-to-input” “output-path”
```

```
public class WordCount {

public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   job.setInputFormatClass(TextInputFormat.class)
   job.setOutputFormatClass(TextOutputFormat.class)
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
//Mapper
public static class TokenizerMapper
     extends Mapper<LongWritable, Text, Text, IntWritable>{
…..
}
//Reducer
public static class IntSumReducer
     extends Reducer<Text, IntWritable, Text, IntWritable> {
…..
}
}
```

Writables

- Hadoop writers for serializing/deserializing input/output data
- Hadoop has implementation for basic Java types
    - int —> IntWritable
    - long —> LongWritable
    - float —> FloatWritable
    - double —> DoubleWritable
    - Boolean —> BooleanWritable
    - String —> Text

Summary of the Hadoop MapReduce job

- InputFormat
  - get the input data process into key-value pairs for the mapper
- Mapper
  - process each key-value pair & produce intermediate key-value pairs
- Combiner
  - an instance of the reducer that run as part of the map task to reduce intermediate output
- Partitioner
  - which key go to which reducer
- Reducer
  - take a key and all its values, process, and produce final output
- OutputFormat
  - write final output from reducers to HDFS

References

- Google Research Paper
  - MapReduce: Simplified Data Processing on Large Clusters
- Apache Hadoop:
  - http://hadoop.apache.org/
- Book
  - Hadoop The Definitive Guide