Java API to MapReduce

Igor Yakushin
ivy2@uchicago.edu

MapReduce: using Java API

- Import various Hadoop related modules:

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

MapReduce: using Java API

- Inside your own WordCount class, create a class that extends Mapper:

```java
public class WordCount {

  public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

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

      public void map(Object 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);
          }
      }
  }
}
```

- TokenizerMapper overwrites map method to split each line into words and return $(word, 1)$ for each $word$.

MapReduce: using Java API

- IntSumReducer extends Reducer class and overwrites its reduce method to count number of words:

```java
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    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);
    }
}
```

MapReduce: using Java API

- main function sets up configuration, launches MapReduce job and writes the results to a file:

```
    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);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

MapReduce: using Java API

```
source env.sh
mkdir wordcount_classes
javac -cp /opt/cloudera/parcels/CDH/lib/hadoop/*:\
          /opt/cloudera/parcels/CDH/lib/hadoop/client-0.20/*\
         -d wordcount_classes WordCount.java
jar -cvf wordcount.jar -C wordcount_classes/ .

hdfs dfs -mkdir /user/$USER/wordcount
hdfs dfs -rm -r -f /user/$USER/wordcount/output
hdfs dfs -put  /software/matlab-2014b-x86_64/\
               toolbox/distcomp/examples/integration/old/pbs/README
               /user/$USER/wordcount/

hadoop jar wordcount.jar WordCount
           /user/$USER/wordcount/README /user/$USER/wordcount/output

hdfs dfs -ls /user/ivy2/wordcount/output
hdfs dfs -cat wordcount/output/part-r-00000
hdfs dfs -cat wordcount/output/part-r-000* | sort > out.txt
hdfs dfs -getmerge wordcount/output merged.txt
cat merged.txt | grep sort
```