Creating a sample Mapreduce

From Distributed Data Mining Winter Semester 2015-2016

Contents

Download a Sample Dataset and deploy in hadoop

Download a Sample dataset. We have use :

```
http://www.gutenberg.org/ebooks/4300.txt.utf-8
```

Deploy the dataset in hadoop cluster:

```
Hadoop fs -put 4300.txt
```

Verify it by:

```
Hadoop fs -ls
```

or by using the browse functionality of your web interface.

```
http://master:50070/explorer.html#/
```

If you are having problems with the Exception ""... bad connect ack with firstBadLink ..."", the first thing you should check is, that you opened the ports **8042** and **50070** on every slave node

```
iptables -I INPUT -i eth0 -p tcp --dport 8042 -j ACCEPT
iptables -I INPUT -i eth0 -p tcp --dport 50010 -j ACCEPT
```

If that doesn't resolve the problem you can try to add the following property to all your hdfs-site.xml [1] (https://www.quora.com/How-should-one-solve-Bad-connect-ack-with-firstBadLink-DataNode-problem-in-Hadoop)

```
<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>8192</value>
</property>
```

Another possible reason could be, that there is not enough space left on the device. Make sure, that you have chosen an image with a 20GB disk or simply restart the filesystem.

```
stop-dfs.sh; stop-yarn.sh; start-dfs.sh; start-yarn.sh
```

Mapper with Python

```
import sys
```

```
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Reducer with Python

```
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
```

```
# input comes from STDIN
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()
   # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
      try:
        count = int(count)
      except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue
    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
        if current_word == word:
           current_count += count
        else:
         if current_word:
            # write result to STDOUT
           print '%s\t%s' % (current_word, current_count)
       current_count = count
       current_word = word
     # do not forget to output the last word if needed!
     if current_word == word:
         print '%s\t%s' % (current_word, current_count)
```

Create a Mapper for Wordcount

Create a mapper code (we have use python) and verify it by testing

```
echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py
```

Create a Reducer for Wordcount

Create a reducer code in your preferred language and verify it via.

```
echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py | sort -k1,1 | /home/hduser/reducer.py
```

Deploy a Dataset in mapreduce

Run you dataset through mapReduce code via

```
hadoop jar /home/hduser/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.6.2.jar -file /home/hduser/mapreduce/mapper.p
```

```
-file /home/hduser/mapreduce/reducer.py -reducer /home/hduser/maprdeuce/reducer.py -input /user/hduser/4300.txt -outpu
```

Mapper with Java

You can use the precompiled version of WordCount in hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.2.jar or use the source code directly and modify it([2] (https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)).

```
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;

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);
     }
   }
 }

 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);
   }
 }

 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);
 }
```

}

For example if you want to remove all chars which are not in the alphabet from the text tokens you can modify the mapper to:

```
public void map(Object key, Text value, Context context
                   ) throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
       String token = itr.nextToken().replaceAll("[^a-zA-Z]", "");
       word.set(token);
       context.write(word, one);
     }
}
```

Then you need to compile the source-code and add it to a jar:

```
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wc.jar WordCount*.class
```

The next step is executing it:

```
/hadoop jar wc.jar WordCount input /output
```

Note, that your input can also be a directory containing multiple single files, but it will throw an Exception if there is a directory in it.

You can now either get the result out of hadoop with

```
hadoop fs -get /output
```

or simple look at it with

```
hadoop fs -cat /output/part-r-00000
```

The result then looks something like this:

```
shorter 1
shortest        1
shortly 2
should          10
show            2
showing 1
shown           1
shows           1
showy           1
shuffled        1
sign            2
signature       1
...
```

Sorting it by the number of occurences, however, can be pretty tricky and needs to be done in a second step, because the output of the reduce step will not be sorted ([3] (http://stackoverflow.com/questions/12343492/mapreduce-how-sort-reduce-output-by-value)).

Verify

Check the output file to verify the output. you can also check the status the status of your running job via WebUI:

```
http://10.155.208.34:8088/cluster
```

Retrieved from "https://i12r-studfilesrv.informatik.tu-muenchen.de/ddmlabws2015/index.php?title=Creating_a_sample_Mapreduce&oldid=4725"