Getting Started with Hadoop

This is a short tutorial on using Hadoop.

We'll go through the process of compiling, packaging, and running a simple Hadoop program.

This tutorial is adapted from the Hadoop Map/Reduce tutorial [https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html]

Logging In

First, make sure you can log in to the head node with SSH, currently at zoidberg.cs.ndsu.nodak.edu. You can log in to this server with your CS Domain password or your Blackboard password.

If you have trouble logging in:

- Check to see if your password works in the Linux lab
- If you **can not** log in to the Linux lab, contact support@cs.ndsu.edu [mailto:support@cs.ndsu.edu]

To request access to the Hadoop cluster, contact support@cs.ndsu.edu [mailto:support@cs.ndsu.edu].

Compiling the WordCount Program

The WordCount program is a simple map reduce program designed to count the number of occurrences of a word given a directory of input files.

This source is from the Hadoop WordCount example.

WordCount.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;

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);
  }
}
```

Put the source file in your home directory on the head node in a file called 'WordCount.java'. You can then compile it on the head node.

From your home directory (where WordCount.java should be), run the following commands (using your username where needed)

```
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wc.jar WordCount*.class
```

This will compile the WordCount.java file and put all of the classes into wc.jar in your home directory.

Setting Up Input Files

This program can use the Hadoop Distributed File System (HDFS) that is set up in the CS department. This file system spans all the Linux lab machines and provides distributed storage for use specifically with Hadoop.

You can work with HDFS with UNIX-like file commands. The list of file commands can be found here [https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html].

First, make a directory to store the input for the program (use your username).

```
helsene@zoidberg:~$ hadoop fs -mkdir /user/helsene/wordcount
helsene@zoidberg:~$ hadoop fs -mkdir /user/helsene/wordcount/input
```

To set up input for the WordCount program, create two files as follows:

file01:

```
Hello World Bye World
```

file02:

```
Hello Hadoop Goodbye Hadoop
```

Save these to your home folder on the head node. To move them into HDFS, use the following commands:

```
helsene@zoidberg:~$ hadoop fs -copyFromLocal /home/helsene/file01 /user/helsene/wordcount/input/file01
helsene@zoidberg:~$ hadoop fs -copyFromLocal /home/helsene/file02 /user/helsene/wordcount/input/file02
```

Again, use your username where applicable.

The syntax here is "hadoop fs -copyFromLocal <LOCAL_FILE> <HDFS_FILE>", in this case we're going to copy file01 from the local system into HDFS under our HDFS user directory into the wordcount/input/ directory.

Running the WordCount Program

You can now run the WordCount program using the following command:

```
hadoop jar wc.jar WordCount /user/username/wordcount/input /user/username/wordcount/output
```

The command syntax is: "hadoop jar <JARFILE> <CLASS> <PARAMETERS…>"

In this case, we use the wc.jar JAR file, running the class 'WordCount' with two parameters, an input directory and an output directory. The output directory must not already exist in HDFS, it will be created by the program.

View Output

You can check the output directory with:

```
hadoop fs -ls /user/username/wordcount/output/
```

You should then see something similar to:

```
helsene@zoidberg:~$ hadoop fs -ls /user/helsene/wordcount/output
Found 2 items
-rw-r--r--   3 helsene nogroup         41 2011-11-08 11:23 /user/helsene/wordcount/output/part-r-00000
```

The 'part-r-00000' file contains the results of the word counting. You can look at the file using the 'cat' command.

```
helsene@zoidberg:~$ hadoop fs -cat /user/helsene/wordcount/output/part-r-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```