CS242 – Information Retrieval & Web Search – Hadoop Cluster

Nhat Le

January 2018

1 Access and Setup Your Personal Hadoop Cluster

Each student is assigned to a lab machine such as `wch13X-YY`. Please remember that our Hadoop clusters run in Pseudo-Distributed mode, thus only work in you assigned machine.

1.1 Setup Hadoop on your assigned lab computer

- First, follow the standard Hadoop instruction provided by our department system group at:
  https://sites.google.com/a/ucr.edu/cse-instructional-support/home/hadoop
  Remember that you need your UCR account to access this document.
- **Fix a small configuration issue caused by our department Hadoop automatic script:** add the following line into the file `$HADOOP_HOME/libexec/hadoop-config.sh`, right before line 167 "Attempt to set JAVA_HOME if it is not set":

```
    export JAVA_HOME=/usr
```

  Now that file will look like:

```
165 export MALLOC_ARENA_MAX=${MALLOC_ARENA_MAX:-4}
166
167 export JAVA_HOME=/usr
168 # Attempt to set JAVA_HOME if it is not set
169 if [[ -z $JAVA_HOME ]]; then
170   # On OSX use java_home (or /Library for older versions)
```

1.2 Run Hadoop commands on your assigned lab computer

All Hadoop commands are in your PATH. So you do not need to type out the path to individual commands. Be aware that lab computers are subject to being rebooted for updates, and also by students using the computers. So, ensure that you save your work and don't procrastinate until right before a deadline.

You can start Hadoop by running following commands in terminal:

- Format filesystem (HDFS):

  ```
  $ hdfs namenode -format
  ```

  This command should be run only once when you start Hadoop for the first time. Otherwise it will erase all data in HDFS.
- Start NameNode deamon and Data daemon:

  ```
  $ start-dfs.sh
  ```
- Start ResourceManager daemon and NodeManager daemon:

  ```
  $ start-yarn.sh
  ```

At this moment, Hadoop is running. You can shutdown it by the following commands:

- `$ stop-yarn.sh`
- `$ stop-dfs.sh`

2 Typical example: WordCount on Hadoop

This section will help you familiar with Hadoop by walking-through a typical example WordCount. Basically, WordCount takes a list of text files as input and output the number of occurrences of each distinct word. We first prepare necessary source code in **your local filesystem**.

- Given that you're in your home directory, try

  ```
  $ mkdir wc
  ```

  ```
  $ cd wc
  ```

- Create `WordCount.java` with the code in Figure 1:
  - `$ touch wc/WordCount.java`
  - Copy the source code in Figure 1 into the created file `wc/WordCount.java`
- Compile `WordCount.java`:
  - Update `HADOOP_CLASSPATH` variable by the following commands:

    ```
    $ export HADOOP_CLASSPATH=$(hadoop classpath)
    $ echo $HADOOP_CLASSPATH
    ```

  - Now compile and create a *jar*:

    ```
    $ javac -classpath $HADOOP_CLASSPATH WordCount.java
    $ jar cf wc.jar WordCount*.class
    ```

The next step is to prepare the input for this example. Remember that both input and output of MapReduce job will be stored on **HDFS**. We will utilize the `xml` files in Hadoop home directory as our input files:

```
$ hdfs dfs -mkdir -p /user/<your_username>/wordcount/input
$ hdfs dfs -put $HADOOP_HOME/etc/hadoop/*.xml /user/<your_username>/wordcount/input
```

Assuming that:

- `/user/<your_username>/wordcount/input` – input directory in HDFS
- `/user/<your_username>/wordcount/output` – output directory in HDFS

Then, we are ready to run WordCount:

```
$ hadoop jar wc.jar WordCount /user/<your_username>/wordcount/input /user/<your_username>/wordcount/output
```

If the job is success, then you can check the result by:

```
$ hdfs dfs -tail /user/<your_username>/wordcount/output/part-r-00000
```

Or even copy the output from HDFS to your local filesystem:

```
$ hdfs dfs -get /user/<your_username>/wordcount/output ~/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);
  }
}
```

Figure 1: WordCount.java

3 Notes

Using 3rd libraries: Add the path of you 3rd libs into `$HADOOP_CLASSPATH` or specify it explicitly with `javac -classpath`.

You will need to do similar thing when running `jar` file. Particularly, you will need something like:

```
$ hadoop jar wc.jar WordCount -libjars mylib.jar
```

Some common HDFS commands:

```
$ hdfs dfs mkdir
$ hdfs dfs put <localsrc> ... <dst> # Copy from local file system to the destination file system
$ hdfs dfs get <src> <localdst> # Copy files to the local file system
$ hdfs dfs cat URI [URI ...] # Copies source paths to stdout.
$ hdfs dfs rm
```

Transfer data between your local machine and Hadoop cluster:

- One of the way is transfering via `bolt` system using `scp, winscp` or any similar tools. Remember that `bolt` limit each student to 400-500MB, thus you need to break your data into smaller pieces.
- Another way is to upload your data to a cloud service and get a sharable link. Then, use this link to download to your Hadoop machine using tools such as `wget`.

Last note:

- Be aware of disk full that used to happen to previous year students (even though, it's rare). Check disk usage by "`df -h /extra/<your_username>`".
- remember that the user directory on `wch13X-YY` is on memory. It is subject to your logout or other student reboot. Please be proactive with your backup, file management, and don't procrastinate until right before a deadline.

4 References

This instruction is based on information from our department instruction for initializing Hadoop, and official tutorial from Hadoop.

- http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html
- http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html
- https://www.google.com/
- Stackoverflow