Introduction

For CS 172, you are provided with access to clusters running Hadoop. Each group in the course has access to one cluster, and you will use this cluster for your coursework. The remainder of this document describes how to access the cluster, what software is installed on it, and how to use that software.

Quick References

How to Access the Cluster

Each group of students has access to one cluster with two nodes in it. You can access your group's cluster using the following steps.

(1) Log in to bolt.cs.ucr.edu using your CS account. You can log in either over SSH (command line client) or X2Go (for a graphical desktop). If you have not done so before, use the following reference:

https://sites.google.com/a/ucr.edu/cse-instructional-support/home/accounts#remote_login

(2) Run the following command to log in to the first node in your group's cluster:

```
cs172_login 1
```

This uses a custom configuration for SSH to access node 1 of your cluster. The same command can be used to access node 2 by changing the argument:

```
cs172_login 2
```

The account you have on the node you are logging in to will be named 'hadoop1', 'hadoop2', or something similar, and is unique to your group’s cluster. The storage in that directory is persistent, and it is shared across all of the nodes in your cluster. So, if you log in to node 1 or 2 in your cluster, you will see the same files when you type 'ls'. All group members share access to the same account on the cluster.

If you want to copy files between bolt.cs.ucr.edu and your cluster, the simplest way to do so is to run the cs172_scp command on bolt.cs.ucr.edu. Here are some examples of doing so:

Example 1: Copy a directory from bolt to your cluster

The following command recursively copies the directory 'my_directory' on bolt.cs.ucr.edu to the top level of your directory on the first node in your cluster.

```
cs172_scp -r my_directory 1:
```

Example 2: Copy a file from bolt to your cluster

The following command copies the file 'my_file.txt' on bolt.cs.ucr.edu to '~/some/directory/my_new_file.txt' on node 1 of the cluster.

```
cs172_scp my_file.txt 1:some/directory/my_new_file.txt
```

Example 3: Copy a directory from the cluster to your account on bolt

The following command recursively copies the directory '~/my_results' on the first node of the cluster to the current directory you are in on bolt.cs.ucr.edu.

```
cs172_scp -r 1:my_results .
```

Example 4: Copy a file from the cluster to your account on bolt

The following command copies the file '~/some/directory/myfile.txt' from your account on the cluster to '~/newfile.txt' in your account on bolt.cs.ucr.edu.

```
cs172_scp 1:some/directory/myfile.txt ~/newfile.txt
```

Hadoop: Storing Files and Running MapReduce

Hadoop has two functions in the context of the cluster - storing files, and running MapReduce jobs. Hadoop's distributed file system, HDFS, can be accessed either by running the command line utility 'hdfs', programmatically using Java code, or code in other languages. Below are some examples of using the hdfs utility to become familiar with HDFS. A full reference of dfs commands is available here:

https://hadoop.apache.org/docs/r2.7.7/hadoop-project-dist/hadoop-common/FileSystemShell.html

```
# List the root directory of your HDFS filesystem.  It should initially be empty!
hdfs dfs -ls /

# Make a directory in hdfs
hdfs dfs -mkdir /example
```

```
# List to see the directory you just made
hdfs dfs -ls /

# Create a file and copy it into the /example directory in hdfs
echo 'This is a test' > afile.txt
hdfs dfs -copyFromLocal afile.txt /example/afile.txt

# Show the contents of that file
hdfs dfs -cat /example/afile.txt

# To see the status of the nodes in your cluster
hdfs dfsadmin -report
```

An example of running a MapReduce job in Hadoop is provided in Appendix: Hadoop MapReduce WordCount

What Can Go Wrong and How to Get Help

Problems that you might encounter would be in one of several categories:

(A) A problem with the cluster itself.
(B) A problem with example code you downloaded and are trying to use.
(C) A problem with your code.

If you encounter a problem that you think might be with the cluster itself, the first thing to do is run the relevant example code from the examples included in this document to see whether the cluster is working or not. For example, if you are having trouble storing data in HDFS in your code, then use 'hdfs dfs' example commands provided above to test. If they work right, then it isn't likely to be a problem with the cluster per se. If they fail (crash, time out, …) then please report the problem immediately to systems@cs.ucr.edu and CC the course instructor and TA, and provide the commands you ran and the error messages you encountered.

One topic of particular relevance is downloading and running examples from the internet, and example code from third parties in general. Even assuming that these online examples worked for the cluster configuration and version of Hadoop that they were originally written for, there will potentially be issues. This is because each cluster is configured differently, and *all* of the software involved keeps changing over time. So, you might run into situations where an example from 2 years ago that worked with older versions of Hadoop is now deprecated and doesn't work on the modern versions. Or where the example code makes hardcoded assumptions about the cluster configuration. In all of these cases, you will need to exercise your best judgement in incorporating such code and features.

Lastly, if you encounter issues where your code isn't working, but the cluster is (see the test examples in this document), then you'll need to debug the issue.

Appendix: Hadoop MapReduce WordCount

Other than just storing files, one of the main functions of Hadoop is to run MapReduce jobs. The following example shows you how to run an example MapReduce job that counts the number of occurrences of words in a file using Java code.

First, download the WordCount.java example provided here onto either your local computer or a session on bolt.cs.ucr.edu.

This file needs to be copied to the cluster and compiled and run there. From bolt.cs.ucr.edu, you would run the following command to copy it over onto the cluster:

```
cs172_scp WordCount.java 1:
```

Once on the cluster, you would run these commands to compile and run the MapReduce job:

```
# Create a directory for the WordCount example
mkdir WordCount
```

Another option is just to gitclone my repo to obtain the files:

```
git clone https://github.com/msalloum/WordCount.git
```

```
# Compile WordCount.java, using the same Java Classpath as Hadoop itself
javac -cp ${HADOOP_CLASSPATH} -d WordCount/ WordCount.java

# Create a Jar file from the Java class files in WordCount
jar -cvf WordCount.jar -C WordCount/ .

# Create a file 'input.txt' that has all of the content before the line that starts with 'End-of-message'
cat > input.txt <<End-of-message
This is line 1 of the message. Bob wrote it.
This is line 2 of the message. Alice wrote it.
This is line 3 of the message. Mallory wrote it.
This is line 4 of the message. Someone else wrote it.
This is the last line of the message.
End-of-message

# Verify that the file has all the content in it that you expect
cat input.txt
```

```
# Create a directory on HDFS for the wordcount example
hdfs dfs -mkdir /wordcount

# Copy the input.txt into HDFS
hdfs dfs -copyFromLocal input.txt /wordcount/input

# Run the MapReduce Job!  /wordcount/input is the input you just created.  /wordcount/output is
the location in HDFS where the output from the job will go
hadoop jar WordCount.jar WordCount /wordcount/input /wordcount/output

# Copy the output from HDFS to your local directory
hdfs dfs -copyToLocal /wordcount/output output

# Look in the local directory where you copied the output.  You should see one or more files.
ls output

# Look at the files you saw.  What is in them?
cat output/<name_of_a_file>
```