Programming Assignment 2: MapReduce on Cloudlab

Due 03/31/2018 midnight

In this assignment, you will program the MapReduce parallel data processing system on the Cloudlab cloud computing platform. This will allow you gain practical experience on MapReduce programming and learn the performance implications of parallel data processing. Be sure not to start too many virtual machine instances to exceed the resource limits for the programming experiments. After you carry out one experiment, it is recommended that you shut down the instances if you do not plan to carry out additional experiments in short time.

Step 1: Hadoop System Structure:

A Hadoop cluster consists of two parallel systems: the Hadoop Distributed File System (HDFS) and the MapReduce framework. Each system performs a different task, but they work together to process large amounts of data. HDFS stores files in a cluster, spread them so they can contain files that are larger than any individual node could store. MapReduce has programs for both the map function and the reduce function, and supports scheduling the tasks in such a way that parallel execution of programs is made possible.

To get started, you need to read the tutorial of MapReduce, available at https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html , to get an idea on how the technical details are pieced together. The tutorial provides an excellent starting point for understanding the general structure of a program and MapReduce as a template to start their own programs. Please note the code “WordCount v2.0” at the bottom contains many useful features that you want to take a look at.

Hadoop API (http://hadoop.apache.org/docs/stable/api/) is the place to find detailed information on the functions. Other information is also available from the official website of Hadoop (http://hadoop.apache.org/).

Step 2: Setup cloudlab account:

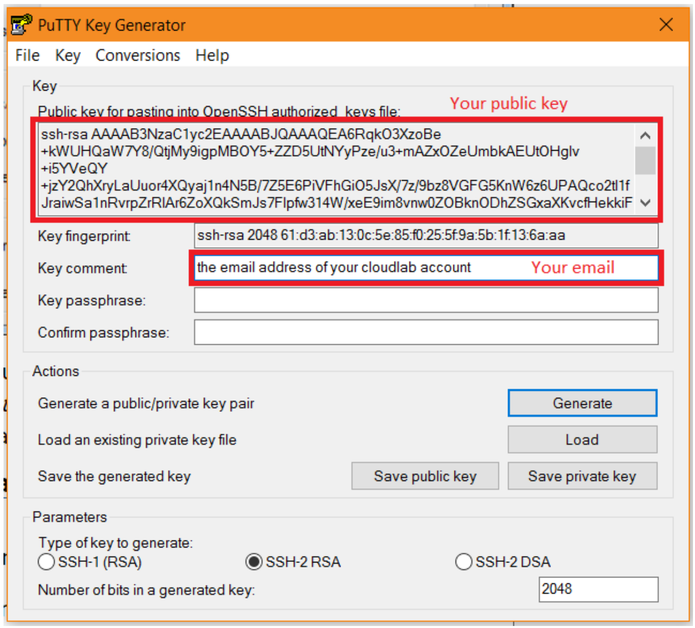
1. Prerequisites:

You can create a new account through https://www.cloudlab.us/login.php
The project we have created is called “EducationProject”, and once you are added to an active project, you can create experiments. During sign up, please choose the option “join existing project” with this project name so that your account can be approved.

2. Add an SSH key to your account

After you logged in your account, click Actions -> Manage SSH Keys
For the add key, there is a public key blank need you to fill in. To generate the key, we use puttygen as an example. You can download it from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

3. Generate a private key (we use puttygen as an example), change the key comment to the email address on your cloudlab account and save it as the private key. Meanwhile copy the public key part for later use.



Public key for pasting into OpenSSH authorized_keys file:

Your public key

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEA6RqkO3XzoBe
+kWUHQaW7Y8/QtjMy9igpMBOY5+ZZD5UtNYyPze/u3+mAZxOZeUmbkAEUtOHglv
+i5YVeQY
+jzY2QhXryLaUuor4XQyaj1n4N5B/7Z5E6PiVFhGiO5JsX/7z/9bz8VGFG5KnW6z6UPAQco2tl1f
JraiwSa1nRvrpZrRlAr6ZoXQkSmJs7Flpfw314W/xeE9im8vnw0ZOBknODhZSGxaXKvcfHekkiF
```

Key fingerprint: ssh-rsa 2048 61:d3:ab:13:0c:5e:85:f0:25:5f:9a:5b:1f:13:6a:aa

Key comment: the email address of your cloudlab account

Your email

Key passphrase:

Confirm passphrase:

Generate a public/private key pair

Load an existing private key file

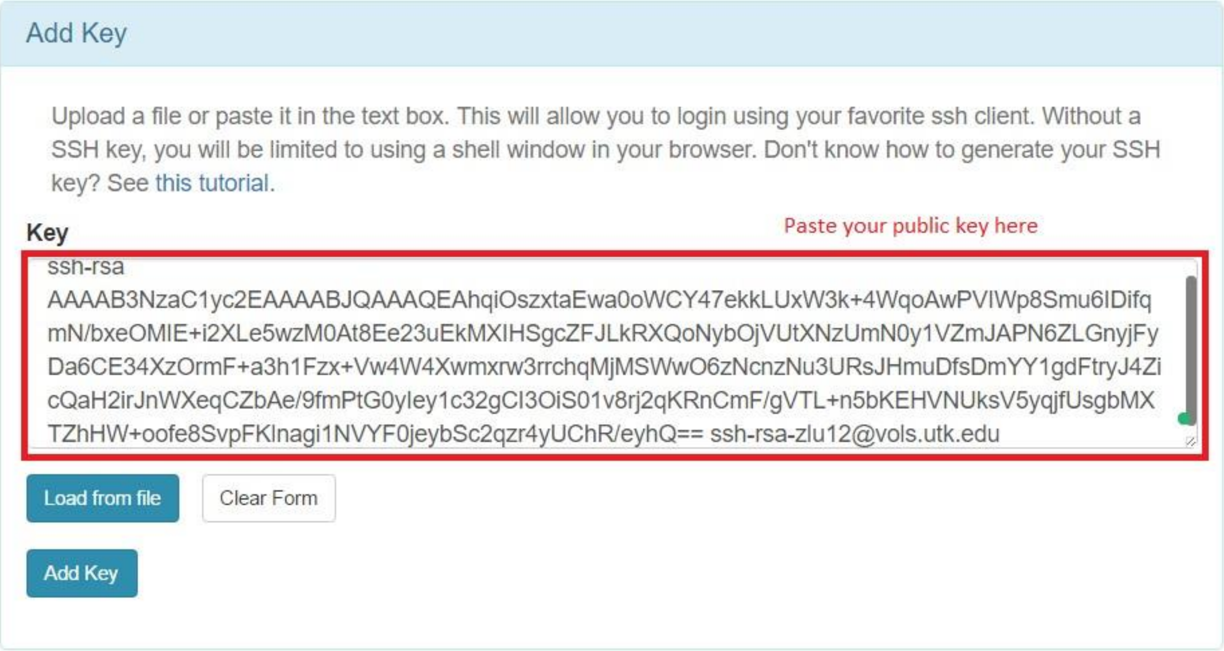
Save the generated key

Type of key to generate:

○ SSH-1 (RSA) ● SSH-2 RSA ○ SSH-2 DSA

Number of bits in a generated key: 2048

4. Paste the public key to fill in the blank in step 1 and click add key



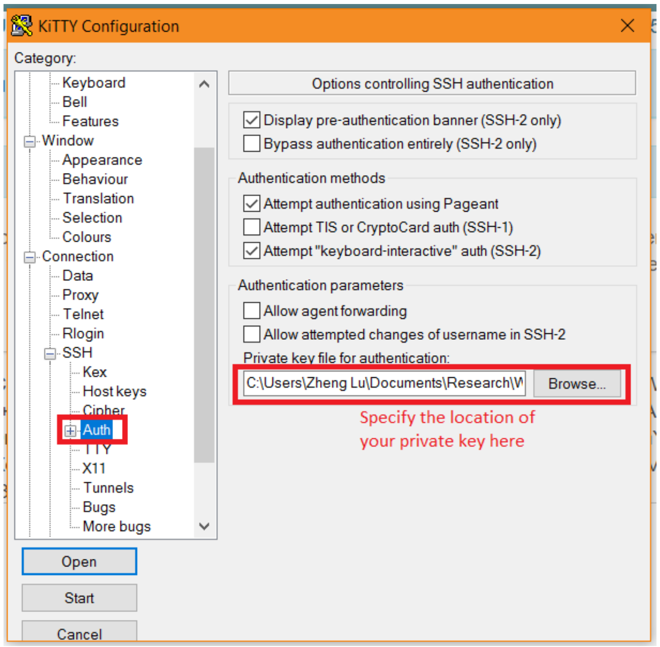
Upload a file or paste it in the text box. This will allow you to login using your favorite ssh client. Without a SSH key, you will be limited to using a shell window in your browser. Don't know how to generate your SSH key? See this tutorial.

Key

Paste your public key here

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEAhqiOszxtaEwa0oWCY47ekkLUxW3k+4WqoAwPVIWp8Smu6IDifq
mN/bxeOMIE+i2XLe5wzM0At8Ee23uEkMXIHSgcZFJLkRXQoNybOjVUtXNzUmN0y1VZmJAPN6ZLGnyjFy
Da6CE34XzOrmF+a3h1Fzx+Vw4W4Xwmxrw3rrchqMjMSWwO6zNcnzNu3URsJHmuDfsDmYY1gdFtryJ4Zi
cQaH2irJnWXeqCZbAe/9fmPtG0yIey1c32gCI3OiS01v8rj2qKRnCmF/gVTL+n5bKEHVNUksV5yqjfUsgbMX
TZhHW+oofe8SvpFKlnagi1NVYF0jeybSc2qzr4yUChR/eyhQ== ssh-rsa-zlu12@vols.utk.edu
```

5. In the future when you need to access nodes in your experiments, use the private key you generated (for putty, specify your key file in SSH->AUTH as shown in following figure). This key should be valid for all your future experiments, you don't have to regenerate it for each experiment.



Options controlling SSH authentication

☑ Display pre-authentication banner (SSH-2 only)
□ Bypass authentication entirely (SSH-2 only)

Authentication methods

☑ Attempt authentication using Pageant
□ Attempt TIS or CryptoCard auth (SSH-1)
☑ Attempt "keyboard-interactive" auth (SSH-2)

Authentication parameters

□ Allow agent forwarding
□ Allow attempted changes of username in SSH-2

Private key file for authentication:
C:\Users\Zheng Lu\Documents\Research\W Browse...

Specify the location of
your private key here

Step 3: Creating a Hadoop cluster on Cloudlab:

1. Log in your cloudlab account
2. Click Actions -> Start Experiment
3. On "Select a Profile" page:
   a. Click Change Profile and scroll down to Other Profiles and pick your profile. b. You will see the cluster contains one resource manager, one namenode and 3 slaves by default.
   c. Click select and click next.
4. On "Parameterize" page:
   a. Choose the number of slave nodes you desire, by default it's 3.
   b. The "use physical nodes" is left unchecked by default, which means virtual machines will be create as nodes in your cluster rather than physical machines.
   c. Click next once you finish the configuration.
5. On "Finalize" page:
   a. For cluster option, you can select which physical location you want to host your cluster. **We recommend Clemson or Wisconsin. We do NOT recommend Utah sites which are based on ARM architecture that may bring troubles later**. However, they may not always available. You can click on the "Check Cluster Status" to check the availability of each cluster. And you can also click on each name of the cluster in there to learn the specifications of machines in each cluster.

    b. Click finish

6. Now your cluster is booting. Wait around 10 minutes until it says “Your experiment is ready”.
7. When your cluster is ready. Click on the List view where you can find the address for each cluster. Log in the resource manager node **with your private key. (refer to last part in step 2)** You can also connect to other nodes with your private key.
8. The Hadoop is located at: **/usr/local/hadoop-[[VERSION]]** You want to add the following to the PATH to invoke commands without problem:

/usr/local/hadoop-[[VERSION]]
/usr/local/hadoop-[[VERSION]]/bin
/usr/local/hadoop-[[VERSION]]/share/hadoop/tools/lib

Step 4: Building the MapReduce code for the Reverse-indexer

In this programming assignment, your job is to design and implement an important component of an online search engine - the capability for users to search full-text from a document. This assignment involves, for the first step, creating a full inverted index of the contents of the document used as input. Here you are required to use the complete works of Shakespeare as testing input, which is available at:

http://www.gutenberg.org/ebooks/author/65

Your program should support indexing of one or more txt files as input. A full inverted index is a mapping of words to their location (document file and offset) in a set of documents. Most modern search engines utilize some form of a full inverted index to process user-submitted queries. In its most basic form, an inverted index is a simple table which maps words in the documents to some sort of document identifier, while a full inverted index maps words in the documents to some sort of document identifier and offset within the document. For example, if given the following two documents:

Doc1: Hello World

You could construct the following inverted file index:

Hello -> (Doc1, 1) World -> (Doc1, 2)

Here the offset is assumed to be the line number (starting with 1). To use Hadoop, you write two classes: a Mapper and a Reducer. The Mapper class contains a map function, which is called once for each input and outputs any number of intermediate pairs. What code you put in the map function depends on the problem you are trying to solve. Let's start with a short example. Suppose the goal is to create an "index" of a body of text -- we are given a text file, and we want to output a list of words annotated with the line-number at which each word appears. For that problem, an appropriate Map strategy is: for each word in the input, output the pair. For now, we can think of the pairs as a nice linear list, but in reality, the Hadoop process runs in parallel on many machines. Each process has a little part of the overall Map input (called a map shard), and maintains its own local cache of the Map output. For a description of how it really works, you should read the Google paper: "MapReduce: Simplified Data Processing on Large Clusters", which we discuss in the class. After the Map phase produces the intermediate pairs they are efficiently and automatically grouped by key by the

Hadoop system in preparation for the Reduce phase (this grouping is known as the Shuffle phase of a map-reduce). For the above example, that means all pairs with the same keys are grouped together.

The Reducer class contains a reduce function, which is then called once for each key. Each reduce looks at all the values for that key and outputs a "summary" value for that key in the final output. Suppose reduce computes a summary value string made of the line numbers sorted into increasing order, then the output of the Reduce phase on the above pairs will produce the pairs that contain the line numbers for each key. Like Map, Reduce is also run in parallel on a group of machines. Each machine is assigned a subset of the keys to work on (known as a reduce shard), and outputs its results into a separate file.

In the indexer assignment, given an input text, we naturally want to create the Map code to extract one word at a time from the input, and the Reduce code to combine all the data for one word. Specifically, you need to program the Hadoop system to divide the (large) input data set into logical "records" and then calls map() once for each record. It is up to you to decide how to implement the details, but since in this example we want to output pairs, the types will both be Text (a basic string wrapper, with UTF8 support). It is necessary to wrap the more basic types as needed.

For the line indexer problem, the map code takes in a line of text and for each word in the line outputs a string key/value pair. When run on many machines, each mapper gets part of the input - so for example with 100 Mbytes of data on 20 mappers, each mapper would get roughly its own 5 Megabytes of data to go through. On a single mapper, map() is called going through the data in its natural order, from start to finish. The Map phase outputs pairs, but what data makes up the key and value is totally up to the Mapper code. In this case, the Mapper uses each word as a key, so the reduction below ends up with pairs grouped by word.

The reduce() method is called once for each key; the values parameter contains all of the values for that key. The Reduce code looks at all the values and then outputs a single "summary" value. Given all the values for the key, the Reduce code typically iterates over all the values and either concatenates the values together in some way to make a large summary object, or combines and reduces the values in some way to yield a short summary value. In this assignment, the summary should include all locations for a word in documents used as input.

Step 5: Running the MapReduce program on the Hadoop cluster

1. Let's assume your mapper and reducer were implemented in Python and edited on your local machine. For example, you can name the code of mapper as **mapper.py** while that of reducer as **reducer.py**.
2. Both **mapper.py** and **reducer.py** must be on the head node of the cluster before we can run them. The easiest way to upload them is to use scp (pscp if you are using a Windows client). From the client, in the same directory as mapper.py and reducer.py, use the following command. Replace **username** with your cloudlab account, and **address** with the address of your cluster.

   ```
   scp mapper.py reducer.py username@address:
   ```

For windows users, you can copy them with winscp (https://winscp.net/eng/download.php). Just follow the similar procedure as we use putty.

Note that you may notice some directories on the servers may have settings that prevent you from uploading files into them. Therefore, you should create your own directory so that you can use winscp to upload your files into them.

3. Connect to the cluster by using putty or SSH:

```
ssh username@address
```

4. Use the following command to start the MapReduce job:

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -files
mapper.py,reducer.py -mapper mapper.py -reducer reducer.py -input
wasb://PATH_TO_INPUT_DATA -output wasb://PATH_TO_OUTPUT
```

5. When the job is complete, use the following command to view the output:

```
hadoop fs -text PATH_TO_OUTPUT/part-00000
```

Deliverables and Grading

Your deliverables include a README file, your source code, and testing result document with some inputs from the works of Shakespeare. Note that we allow any language for the mapper and reducer, such as Python, Java, C#, or others that are supported. Your code should achieve the following:

(20%) Identifying and removing stop words – One issue is that some words are so common that their presence in an inverted index is "noise," that is they can obfuscate the more interesting properties of a document. Such words are called “stop words.” For this part of the assignment, write a word count MapReduce function to perform a word count over a corpus of text files and to identify stop words. It is up to you to choose a reasonable threshold (word count frequency) for stop words, but make sure you provide adequate justification and explanation of your choice. A parser will group words by attributes which are not relevant to their meaning (e.g., "hello", "Hello", and "HELLO" are all the same word), so it is up to you to define "scrub" however you wish; some suggestions include case-insensitivity, etc. It is not required that you treat “run” and “ran” as the same word, but your parser should handle case insensitivity. Once you have written your code, then run your code and collect the word counts for submission with all your Mapper and Reducer files.

Note: you should not directly use any stop word lists that are available online.

(40%) Building the Inverted Index -- For this portion of the assignment, you will design a MapReduce-based algorithm to calculate the inverted index. To this end, you are to create a full inverted index, which maps words to their document ID + line number in the document. Note that your final inverted index should not contain the words identified in Step 1. The format of your MapReduce output (i.e., the inverted index) must be simple enough to be machine-parseable; it is not impossible to imagine your index being one of many data structures used in a search engine's indexing pipeline. Your submitted indexer should be able to run successfully on one or multiple input txt files, where "successfully" means it should run to completion without errors or exceptions, and generate the correct word->DocID mapping. You are required to submit all relevant Mapper and Reducer Java files, in addition to any supporting code or utilities.

(40%) Query the Inverted Index -- Write a query program on top of your full inverted file index that accepts a user-specified query (one or more words) and returns not only the document IDs but also the locations in the form of line numbers. The query program can be local: it does not need to handle the task using Map-Reduce framework again. It is not required that your query program to return text snippets from the original text files.

(Extra credit: 20%) Integrate queries with Spark: In the extra credit, you are required to implement the query operation of the inverted index using Spark framework. Note that given that Spark is interactive, you will need to first load the index from disk using RDDs, and then write queries on these RDDs. You only need to support the search operation, i.e., returning the search hits in the interactive Spark Shell. You need to demonstrate this using screenshots or a video illustrating your Spark shell operations.

Note that this programming assignment, just like the previous one, does not have a single correct solution. You need to make design decisions. So please document them in the README file, and also include detailed instructions on how to use your code. You need to put the sample input and output results into the testing report for the grading purposes.

Resources:

Here are some resources available for making the project easier.

HDFS Basic Commands:

HDFS is a distributed file system and its shell-like commands are close to the typical Linux shell command. The following basic commands are from the official site (http://hadoop.apache.org/docs/r0.18.3/hdfs_shell.html), which is quite useful.

- ls, to list a directory, usage: /usr/local/hadoop-2.7.1/bin/hadoop fs -ls args
- mkdir, to create a new directory or file, usage: /usr/local/hadoop-2.7.1/bin/hadoop fs -mkdir args
- mv, to move a directory or file from one place to another, usage: /usr/local/hadoop-
- 2.7.1/bin/hadoop fs - mv args
- rm, to remove a directory or file, usage: /usr/local/hadoop-2.7.1/bin/hadoop fs -rm args
- cp, to copy a directory or file from one place to another, usage: /usr/local/hadoop-2.7.1/bin/hadoop fs -cp args
- put, to copy from local to HDFS, usage: /usr/local/hadoop-2.7.1/bin/hadoop fs -put args
- get, to copy from HDFS to local, usage: /usr/local/hadoop-2.7.1/bin/hadoop fs -get args

Hadoop word count example:

The following content is based on https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html .

1. Install jdk, in my case, it is installed under /usr/lib/jvm/default-java
2. Set up environments
   ```
   export HADOOP_HOME=/usr/local/hadoop-2.7.1
   export JAVA_HOME=/usr/lib/jvm/default-java
   export PATH=${JAVA_HOME}/bin:${HADOOP_HOME}/bin:${PATH}
   export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
   ```
3. Compile source code:
   ```
   hadoop com.sun.tools.javac.Main WordCount.java
   jar cf wc.jar WordCount*.class
   ```
4. Import input files to HDFS:
   ```
   sudo -s /usr/local/hadoop-2.7.1/bin/hadoop fs -mkdir /test
   sudo -s /usr/local/hadoop-2.7.1/bin/hadoop fs -chmod 777 /test
   hadoop fs -put file01 /test/file01
   hadoop fs -put file02 /test/file02
   hadoop fs -ls /test
   ```
5. Run the program:
   ```
   hadoop jar wc.jar WordCount /test /test/output
   ```
6. Check the result:
   ```
   hadoop fs -cat /test/output/part-r-00000
   ```
7. Done!