
Assignment 5
Due: 11:59:59 PM, April 11 (Tuesday).

The goal of this assignment is to learn two distributed computing frameworks. Hadoop is used in many big data applications to exploit distributed systems. MPI is used in many scientific computing applications.

Question 1 (Hadoop, 60 points)

This assignment should be done in teams of two. The Jar and Java files have to be zipped into a file named `EID1_EID2.zip` and submitted through Canvas. Before the submission, you should make sure that the program is able to run on the latest Hadoop using single node (i.e., pseudo-distributed operation mode).

In this assignment, you will implement a text analyzer using Hadoop, which is a programming model and software framework for developing applications that concurrently process large scale data (Big Data) on distributed systems. The analyzer has to build a table that shows the occurrences of the words that appear together in the given text. The data set for this problem is the novel *Pride and Prejudice*. A collection of files (one per chapter) is provided in the attached zip file for the assignment. The occurrences are calculated in individual lines; your program does not need to intelligently separate sentences because the Mapper function is invoked for each `newline` by default.

Here is an example for calculating the occurrences:

- **Text:** *"Mr. Bingley was good-looking and gentlemanlike; he had a pleasant countenance, and easy, unaffected manners."*
- **Occurrences:** For the word **Bingley**, *"and"* appears twice and *"gentlemanlike"* appears once. In this calculation, we say **Bingley** is a 'contextword'; *"and"* and *"gentlemanlike"* are 'queryword'. The other example: for the contextword **and**, *"pleasant"* appears once, *"and"* appears once (not zero times or twice).

For simplicity of this assignment and obtaining the same results, you should 1) convert every character into lower-case, 2) mask non-word characters by white-space; i.e., any character other than `a-z`, `A-Z`, or `0-9` should be replaced by a single-space character. For instance, "*Bingley's book*" is converted into "*bingley s book*". If the text of the novel is fully analyzed, we should get these results:

1. *Darcy* occurs 112 times with *Bingley*; *Bingley* is the contextword and *Darcy* is the queryword.
2. *Elizabeth* occurs 135 times with *Jane*; *Jane* is the contextword, and *Elizabeth* is the queryword.

Your Hadoop program has to output a table that contains all combinations of different contextwords and querywords. The table should follow the format (and only this format) of:

```
contextword1
<queryword1, occurrence>
<queryword2, occurrence>

contextword2
<queryword1, occurrence>
<queryword2, occurrence>
```

where the querywords for the same contextword are sorted in their lexicographic order.

Quick Guide for Setting Up Hadoop

Hadoop has two main modules: HDFS (Hadoop Distributed File System) and Yarn. The HDFS modules provides the storage for MapReduce tasks and the Yarn module manages the tasks and the resources for processing data in MapReduce framework. In the logical view, the HDFS module has two kinds of node: name node and data node. The name node accepts the requests for accessing/changing the data in the file system and data node performs the operations to the data. In Yarn module, the two nodes are resource manager and node manager. In an oversimplified view, the resource manager accepts the MapReduce tasks and distributes the tasks to different node manager. The node manager is a per-machine agent who monitors the resource usage of that machine.

Before you start working on this assignment, you should read the Tutorial on Hadoop webpage for developing a Hadoop application. In this assignment, we will use Hadoop 2.8.0. The following steps will guide you through the setup of Hadoop 2.8.0 on a Linux machine.
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

The following link is for installing on mac:
https://amodernstory.com/2014/09/23/installing-hadoop-on-mac-osx-yosemite/

DO NOT simply copy and paste the settings, you may need to replace the variables accordingly.

1. Add the following lines to the .bashrc on all machines:

```
export JAVA_HOME=/path/to/jre/directory
export HADOOP_HOME=/path/to/the/hadoop/directory
export HADOOP_MAPRED_HOME=$HADOOP_HOME  # home to MapReduce module
export HADOOP_COMMON_HOME=$HADOOP_HOME  # home to common module
export HADOOP_HDFS_HOME=$HADOOP_HOME    # home to HDFS module
export YARN_HOME=$HADOOP_HOME           # home to Yarn module
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop # path to configurations
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin # add Hadoop binaries to PATH
```

   After adding those lines into your .bashrc, you can use the command "source .bashrc" or re-login to make the changes happen.

2. Create storages for namenode and datanode. In this example, the folders for the storages are located in `$HADOOP_HOME/hdfs`; however, you can put them in any directory you want.

```
mkdir -p $HADOOP_HOME/hdfs/namenode
mkdir -p $HADOOP_HOME/hdfs/datanode
```

3. Setup the configuration files, which are located in `$HADOOP_HOME/etc/hadoop`, for Hadoop and Yarn. Put the following settings between <configuration> and </configuration> in each file.

   (a) yarn-site.xml: we add the default shuffle function into the system.

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
```

```
            <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
    </configuration>
```

(b) core-site.xml: create a single node on the localhost. In this example, we use port 9000 for hadoop. You can use any other available port. Note that, you must ensure that the port you want to use is available.

```
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
```

(c) hdfs-site.xml: specify the storage for HDFS. Note that, `$HADOOP_HOME` should be replaced by **the absolute path**.

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:$HADOOP_HOME/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:$HADOOP_HOME/hdfs/datanode</value>
    </property>
</configuration>
```

(d) mapred-site.xml: this file does not exist at first, so use "`cp mapred-site.xml.template mapred-site.xml`" to create an initial configuration file, then put the setting into the file.

```
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
```

4. Now check that you can ssh to the localhost without a passphrase:

```
  $ ssh localhost
If you cannot ssh to localhost without a passphrase, execute the following commands:

  $ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
  $ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
  $ chmod 0600 ~/.ssh/authorized_keys
  $ chmod og-wx ~/.ssh/authorized_keys (if above chmod doesn't work try this)
```

5. Format the storage for the name node before Hadoop is started.

```
bin/hdfs namenode -format
```

If you see "`INFO util.ExitUtil: Exiting with status 0`" in the last few lines, then you are good. If the returned status is 1, you can check the error message on the screen and fix the problem.

6. Start the hdfs filesystem daemons using the following commands:
   `sbin/start-all.sh`
7. Make the HDFS directories required to execute MapReduce jobs:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/<username>      //use your username
```

8. Check everything is working
   (a) Check the log files in `$HADOOP_HOME/logs`.
   (b) The following webpage should be accessible:

```
http://localhost:50070/ for NameNode
http://localhost:8088/ for ResourceManager
```

   If you are using an ECE machine, you might not be able to access the webpages because of firewall.

   (c) Use "`jps`" command to list the Java daemons in the background. If all the processes are started successfully, you should see something similar to the following example:

```
22837 NodeManager
23465 Jps
22540 ResourceManager
16804 NameNode
17056 SecondaryNameNode
16927 DataNode
```

   Every line starts with a process ID and then followed by the process name.

9. Run the basic Hadoop example.

```
mkdir input  # create a local folder
vim input/file  # create and edit a file. Type down some words in the file.
hdfs dfs -copyFromLocal input input # copy the local directory to HDFS
(or bin/hdfs dfs -put input input)
hdfs dfs -ls input # list the files in the directory input on HDFS
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar
                              wordcount input output # run the famous hadoop example
hdfs dfs -ls output # you should see all the outputs
hdfs dfs -copyToLocal output # download the result from HDFS
(or  hdfs dfs -get output output)
```

10. Stop Hadoop

```
$sbin/stop-all.sh
```

Note: we show the most basic commands for starting and stopping Hadoop for teaching purpose; however, you may use other commands that are more convenient to do the same task.

Compile a Hadoop Application

1. Compile (the command "`hadoop classpath`" returns all the dependencies required by Hadoop):
   `javac -classpath `hadoop classpath` -d build src/TextAnalyzer.java`
   The backtick (`) symbol is the same key as the tilde (~) and is located on the top-left of your keyboard.
2. Jar the Hadoop application: (You can remove the v in -cvf for a clean and quite output)
   `jar -cvf TextAnalyzer.jar -C build/ .`

Run a Hadoop Application

1. Make sure you have uploaded the input files onto HDFS.
2. Use the command to remove the output directory before the next run. Hadoop does not over-written the existing output directory and it stops the task if there exists one.
   ```
   hdfs dfs -rm -r output
   ```
3. Use the command to run your Hadoop application. The fourth field (which is TextAnalyzer in the example) is the class that contains the main method.
   ```
   hadoop jar TextAnalyzer.jar TextAnalyzer input output
   ```

Before submitting your program, you need to make sure that your program is able to run on Hadoop 2.8.0 using pseudo-distributed mode.

Useful Links

1. *MapReduce paper* – http://research.google.com/archive/mapreduce.html
2. *Hadoop* – http://hadoop.apache.org/
3. *Setup Hadoop* – http://hadoop.apache.org/docs/current/index.html
4. *Develop a Hadoop Application* – https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Question 2, MPI 40 points

Write a program using MPI to solve Row-wise Matrix Vector Multiplication. You will be provided with input files **matrix.txt** and **vector.txt** (samples are on canvas). Your result must be output to a file named **result.txt**. You are only allowed to read in the matrix M and vector V from one MPI process. Similarly, you need to output to the result file at a single MPI process.

The source code (C/C++) of the programming part must be uploaded through the canvas before the end of the due date. The assignment should be done in teams of two. Please zip and name the source code as [EID1_EID2].zip.

Inputs:

- Read an NxM Matrix stored in a text file named matrix.txt
- Read a Vector V of size M stored in text file vector.txt

Outputs:

- Write the output vector to file result.txt

Divide the input matrix into roughly equal sized chunks n/p chunks (total rows: n, total processes: p). **You cannot assume that n is divisble by p.** Each process will work on its own chunk and send the result back to the process which is responsible for writing the result.

Resources:

- MPI Tutorials

  {http://mpitutorial.com/tutorials/}

  is an excellent resource on how to write MPI programs.

- How to run MPI programs on Stampede

```
Steps to run MPI programs on Stampede

1. SSH to stampede ssh userid@stampede.tacc.utexas.edu

You mave to finish your two step authentication.

2. compile your c/c++ code such as hello.c by using the following commands
mpicc hello.c -o hello.out
mpic++ hello.cpp -o hello.out

This command should produce hello.out

3. Create a batch file run-batch

#!/bin/bash
#SBATCH -J myMPI          # job name
#SBATCH -o myMPI%j       # output and error file name (%j expands to jobID)
#SBATCH -n 2              # total number of mpi tasks requested
```

```
#SBATCH -p normal     # queue (partition) -- normal, development, etc.
#SBATCH -t 00:00:20        # run time (hh:mm:ss) - 20 seconds
#SBATCH -A EE-360P-Concurrent-a
#SBATCH --mail-user=user@utexas.edu # replace by your email
#SBATCH --mail-type=begin  # email me when the job starts
#SBATCH --mail-type=end    # email me when the job finishes
ibrun  ./a.out
```

Note that '#' should be there as shown above

4. submit the job by running

```
sbatch run-batch
```

It should show a message similar to:

```
------------------------------------------------------------------
              Welcome to the Stampede Supercomputer
------------------------------------------------------------------

No reservation for this job
--> Verifying valid submit host (login4)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/04311/asad)...OK
--> Verifying availability of your work dir (/work/04311/asad)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (normal)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (EE-360P-Concurrent-a)...OK
Submitted batch job 8190878
```

5. The output of your job will appear in a file such as myMPI8190878

6. You can change options in the batch file if you want.
For example, if you do not want email on job completion, you can remove that line.
Similarly, the number of mpi tasks can be changed (this will depend on your assignment)