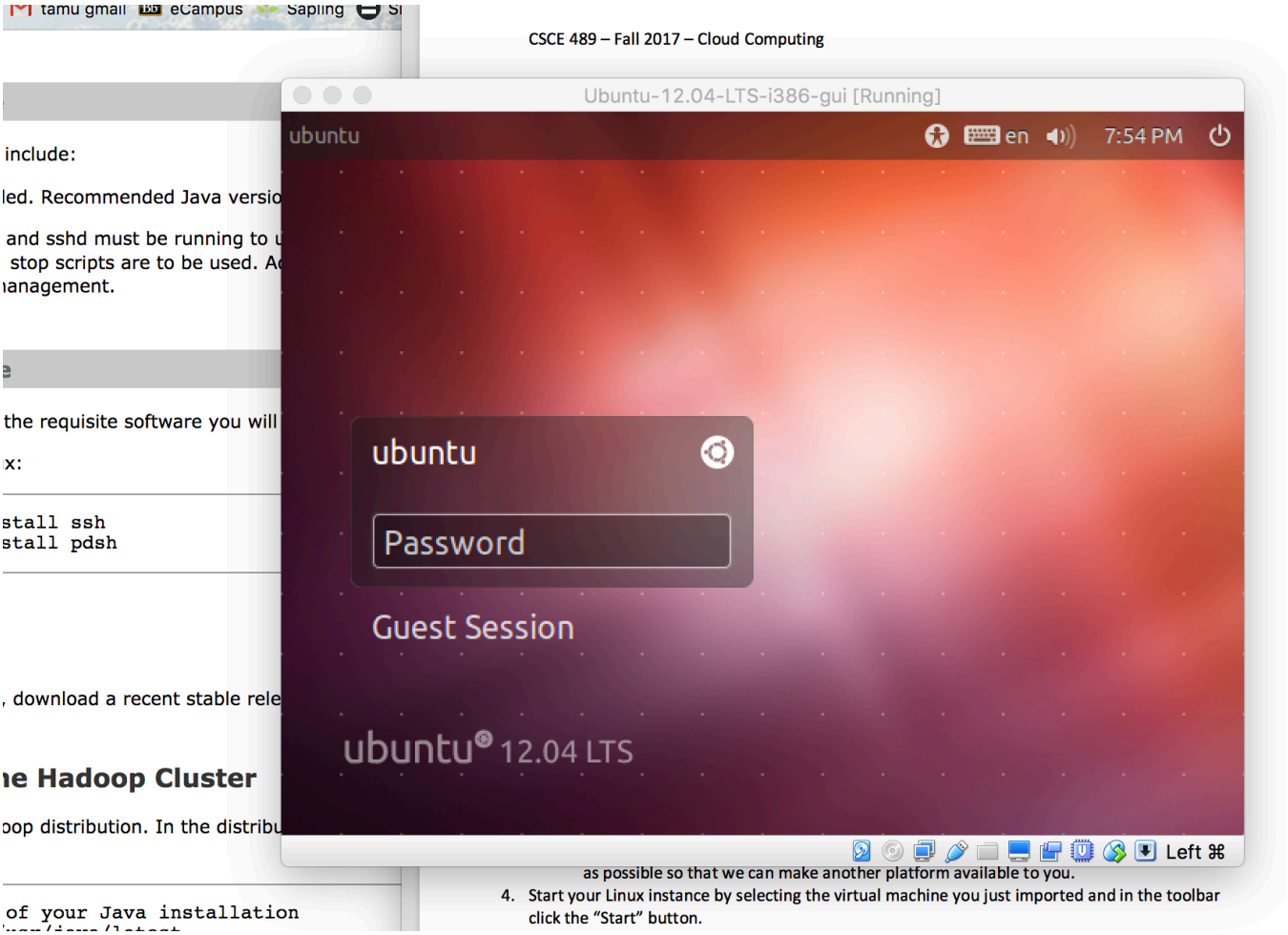
Project3 Report

Nathaniel Leake, 424003778
Time to complete Task: **8hrs**

Step 1: Preparing a Linux Environment

Time Spent: (day1) 6:00pm-8:00, 2 hours

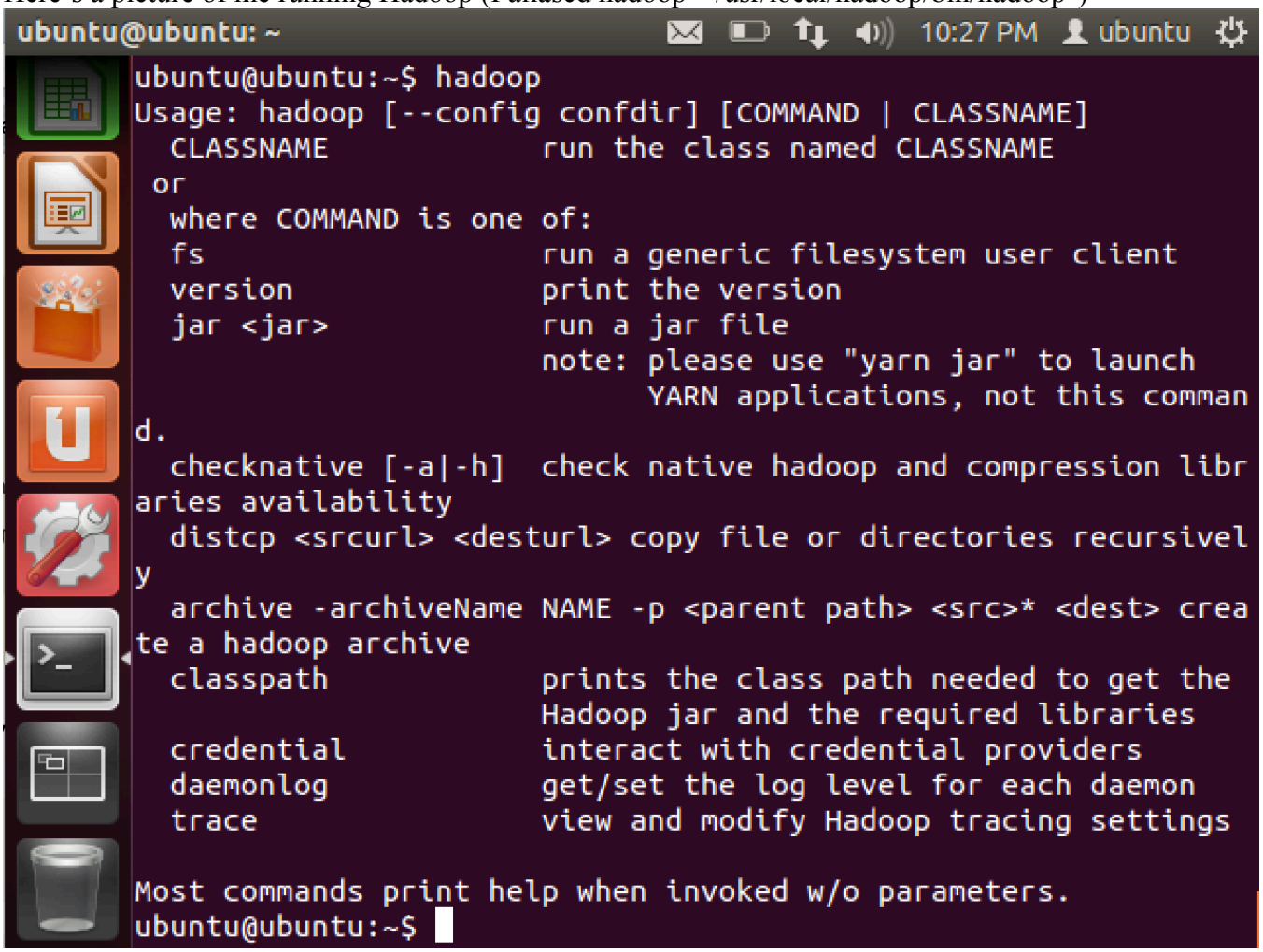
The reason this step took so long was the Ubuntu .ISO download. Due to poor wifi connection at the time, it took over an hour to get this file, during which time I reviewed the project requirements and went over the VirtualBox tutorials so that I was prepared to install the Ubuntu machine once I got the ISO. Getting the machine imported and started required a bit of getting used to VirtualBox, but that wasn't too bad.



Step 2: Install Hadoop

Time Spent: (day2) Getting Root Access=1h, Getting Libraries=30m, Getting Hadoop Running=50m, Total=2.3 hours

I was able to the account with the password "nimda", but the root password "toor" for sudo wasn't working. I tried lots of on online solutions and variations of "root", "toor", and my system password, but in the end I just gave up and downloaded a raw Ubuntu ISO from Ubuntu's main website and set up the configurations myself. After this, the "bin/hadoop" file was not found, so I reinstalled Hadoop, added a lot of different settings, such as the Java path, to various files (as directed by online articles & Apache help pages), and moved directories around until everything was finally working. Here's a picture of me running Hadoop (I aliased hadoop="/usr/local/hadoop/bin/hadoop")



Step 3: Go through Hadoop WordCount Example

Time Spent: 68 minutes (or 1.1 hours)

MapReduce is something I have been studying a lot this semester. In fact, I wrote an essay on it in another class before it was even brought up in the lecture videos for our class. This source code for this program was very cool to look at – Java was my first programing language, and Hadoop libraries look easier to call than I expected. I read through entire example. Below is a screenshot of the final output of the algorithm from the last page:

```
<Hadoop, 3>
<Oh, 1>
<a, 1>
<an, 1>
<as, 2>
<be, 1>
<can, 1>
<elephant, 1>
<fellow, 1>
<is, 3>
<what, 1>
<yellow, 2>
```

Categories: Applications | Hadoop | MapReduce | Tutorial | Walkthrough | WordCount 1.0 | All Categories

Running WordCount v1.0 Example: WordCount v2.0

I also ran this program in Hadoop to watch it perform first-hand:

```
                Merged Map outputs=0
                GC time elapsed (ms)=6
                Total committed heap usage (bytes)=16252928
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=8
ubuntu@ubuntu:~/hadoop_tutorial/WordCount1$ hadoop jar wordcount.ja
r org.myorg.WordCount input/ output/
```

Here's a slightly more complicated example of how to run a sample program in Hadoop that I found under the /etc/ folder in Hadoop's install

```
ubuntu@ubuntu:~$ hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.2.jar grep ~/input ~/grep_example
 'principal[.]*'
```

Step 4: Going through Hadoop Pig Application

Time Spent: 2 hours

Pig Latin is gross. I spent a long time on these scripts, after installing pig. The main hiccups in this project were finding the tutorial script files and making sure I typed everything in correctly from the tutorial website. Here is my work from the second script in pig:

```
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDecector(query);
eanclay1 = ILTERFAY awray YBAY orgway.apacheway.igpay.utorialtay.OnUR
LDecectornay(eryquay);
clean2 = FOREACH clean1 GENERATE user, time, org.apache.pig.tutorial.
ToLower(query) as query
eanclay2 = OREACHFAY eanclay1 ENERATEGAY userway, imetay, orgway.apac
heway.igpay.utorialtay.OLowertay(eryquay) asway eryquay
houred = FOREACH clean2 GENERATE user, org.apache.pig.tutorial.Extrac
tHour(time) as hour, query;
ouredhay = OREACHFAY eanclay2 ENERATEGAY userway, orgway.apacheway.ig
pay.utorialtay.ExtractHourway(imetay) asway ourhay, eryquay;
ngramed1 = FOREACH houred GENERATE user, hour, flatten(org.apache.pig
.tutorial.NGramGenerator(query)) as ngram;
amedngray1 = OREACHFAY ouredhay ENERATEGAY userway, ourhay, attenflay
(orgway.apacheway.igpay.utorialtay.AmGeneratornGray(eryquay)) asway a
mngray;
ngramed2 = DISTINCT ngramed1;
amedngray2 = ISTINCTDAY amedngray1;
hour_frequency1 = GROUP ngramed2 BY (ngram, hour)
ourhay_equencyfray1 = OUPGRAY amedngray2 YBAY (amngray, ourhay)
hour_frequency2 = FOREACH hour_frequency1 GENERATE flatten($0), COUNT
($1) as count
ourhay_equencyfray2 = OREACHFAY ourhay_equencyfray1 ENERATEGAY attenf
lay($0), OUNTCAY($1) asway ountcay
hour_frequency3 = FOREACH hour_frequency2 GENERATE $0 as ngram, $1 as
 hour, $2 as count;
ourhay_equencyfray3 = OREACHFAY ourhay_equencyfray2 ENERATEGAY $0 asw
ay amngray, $1 asway ourhay, $2 asway ountcay;
hours00 = FILTER hour_frequency2 BY hour eq '00';
ourshay00 = ILTERFAY ourhay_equencyfray2 YBAY ourhay eqway '00';
hour12 = FILTER hour_frequency3 BY hour eq '12';
ourhay12 = ILTERFAY ourhay_equencyfray3 YBAY ourhay eqway '12';
same = JOIN hour00 BY $0, hour12 BY $0
amesay = OINJAY ourhay00 YBAY $0, ourhay12 YBAY $0
same1 = FOREACH same GENERATE hour_frequency2::hour00::group::ngram a
amesay1 = OREACHFAY amesay ENERATEGAY ourhay_equencyfray2::ourhay00::
sway ountcay 12;
STORE same1 INTO '/tmp/tutorial-join-results' USING PigStorage();
ORESTAY amesay1 INTOWAY '/tmpay/utorialtay-oinjay-esultsray' USINGWAY
```

I still don't fully understand Pig and why it is built the way it is.

Bonus: Tweet Analyzer

Time Spent: 15 minutes (0.25 hours)

Solving this would be almost as easy as the first Hadoop example program tutorial. We simply need to use the MapReduce API to query for the following structures on our database of users:

- Hashtags in tweets, converted into tuples representing count, and reduced by users from top 50 cities and then fully accumulated/reduced into resulting counts.
- We will want specify URLS of popular newspapers, magazines, and TV shows as paths in the mapping job and reduce based off of word count (occurrence of a word), similarly to the WordCount example but with the mapping restriction and a lot more data to scan over.
- For most popular video, we just a "video URL count" mapping for posted videos that does a search and sort before returning just the most used video URL. This could be made more accurate by only looking at distinct video URLS for a single user to prevent bots from spamming a URL to trick the query into thinking it has a high popularity.

In short, the answers are all in MapReduce.

Total Report Time: [2 + 2.3 + 1.1 + 2] + 0.25 = 7.65 hours