The City College of New York | Department of Computer Science

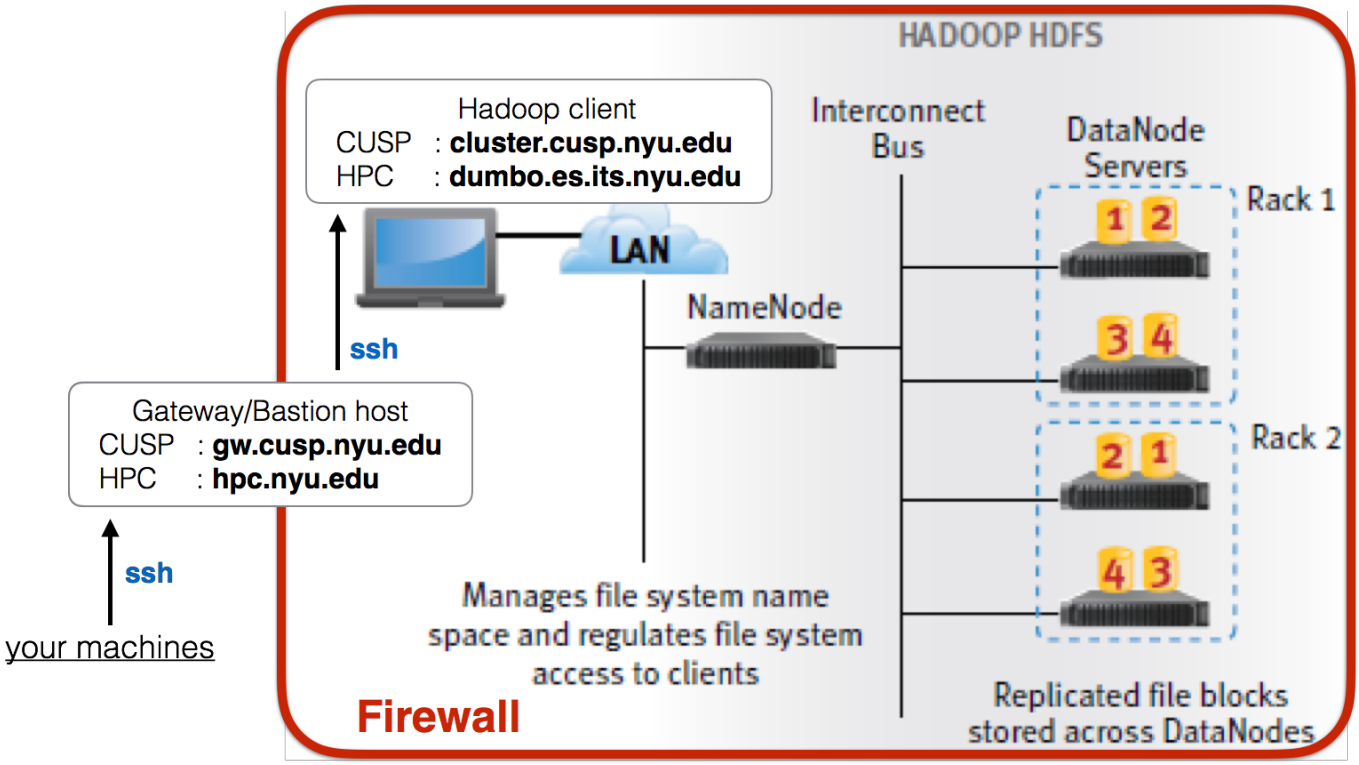
DSE I2450: Big Data and Scalable Computation

SUMMER 2018

Lab 6 – Hadoop @ NYU/CUSP

In this lab, we're going to get familiar with the Hadoop computing environment by running Hadoop Streaming jobs on the clusters at NYU-HPC/CUSP. You should already have access to NYU HPC cluster. You will need a CUSP account for the cluster at NYU-CUSP. Please make sure that you have a Terminal with SSH capability on your machine (e.g. Terminal on Linux/Mac OS X or Putty on Windows).

General Access Diagram for the Hadoop Clusters @ NYU



In general, both CUSP and HPC clusters are protected by a firewall, thus, no direct SSH access are allowed. In order to access the Hadoop clusters, we must first login to a gateway or a bastion host, i.e. **gw.cusp.nyu.edu** and **hpc.nyu.edu** for the CUSP and HPC cluster, respectively. From this gateway, we can then access the Hadoop client machine, **cluster.cusp.nyu.edu** and **dumbo.es.its.nyu.edu**. Both of the machines are equipped with a Hadoop environment for us to develop and run Hadoop jobs on their respective cluster. Note, if you're on NYU-NET, you can connect directly to dumbo.

TASK 1 – Access the Hadoop client node

Before we can perform any Hadoop work, we need to be on the Hadoop client node. In this case, we'll need to be on **cluster.cusp.nyu.edu** or **dumbo.es.its.nyu.edu**. The rest of lab will assume that we're using the CUSP cluster. Specific instructions for the HPC cluster are also provided where applicable. First, let's login into the CUSP cluster by running the following commands in your Terminal (the "$" sign indicates the bash prompt, and thus, only the contents after "$" are part of the commands):

```
local$ ssh <YOUR_NET_ID>@gw.cusp.nyu.edu

Last login: ….
[NetID@gw ~]$ ssh cluster

[NetID@login-1-1 ~]$ which hadoop
/usr/bin/hadoop
```

TASK 2 – Code preparation

Next, we need to clone the GIT repo for this lab:

```
[NetID@login-1-1 ~]$ git clone https://github.com/hvo/mrlab.git
Initialized empty Git repository in mrlab/.git/
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 2), reused 0 (delta 0)
Receiving objects: 100% (10/10), 90.01 KiB, done.
Resolving deltas: 100% (2/2), done.
```

If you get prompted for a host check confirmation of RSA fingerprint, please type 'yes' to continue.

After this, we should see the following contents in the mrlab directory:

```
[NetID@login-1-1 ~]$ cd mrlab
[NetID@login-1-1 mrlab]$ ls -l
-rw-rw-r--. 1 hvo hvo 259337 Mar 23 22:33 book.txt
-rwxrwxr-x. 1 hvo hvo    292 Mar 23 22:33 mapper2.py
-rwxrwxr-x. 1 hvo hvo    292 Mar 23 22:33 mapper.py
-rwxrwxr-x. 1 hvo hvo    409 Mar 23 22:33 reducer2.py
-rwxrwxr-x. 1 hvo hvo    409 Mar 23 22:33 reducer.py
-rwxrwxr-x. 1 hvo hvo    607 Mar 23 22:33 run.sh
-rwxrwxr-x. 1 hvo hvo     73 Mar 23 22:33 test_run.sh
```

TASK 3 – Run the WordCount example with Hadoop Streaming

1. Upload data onto HDFS:

```
[NetID@login-1-1 mrlab]$ hadoop fs -put book.txt .
```

2. You can verify the contents of your HDFS folder by running:

```
[NetID@login-1-1 mrlab]$ hadoop fs -ls
Found 1 item
-rw-------   3 htv210 users     259334 2016-03-01 22:17 book.txt
```

3. Run the WordCount example:

```
[NetID@login-1-1 mrlab]$ ./run.sh mapper.py reducer.py book.txt output 2 2 counts.txt
rm: `output': No such file or directory
16/03/23 22:49:16 INFO Configuration.deprecation: mapred.job.name is deprecated.
Instead, use mapreduce.job.name
16/03/23 22:49:16 INFO Configuration.deprecation: mapred.map.tasks is deprecated.
Instead, use mapreduce.job.maps
streaming-2.6.0-cdh5.4.5.jar] /tmp/streamjob9126574211602756291.jar tmpDir=null
16/03/23 22:49:16 INFO client.ConfiguredRMFailoverProxyProvider: Failing over to
rm356
16/03/23 22:49:17 INFO mapred.FileInputFormat: Total input paths to process : 1
16/03/23 22:49:17 INFO mapreduce.JobSubmitter: number of splits:2
16/03/23 22:49:17 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1455656680779_0161
16/03/23 22:49:18 INFO impl.YarnClientImpl: Submitted application
application_1455656680779_0161
16/03/23 22:49:18 INFO mapreduce.Job: Running job: job_1455656680779_0161
16/03/23 22:49:29 INFO mapreduce.Job: Job job_1455656680779_0161 running in uber mode
: false
16/03/23 22:49:29 INFO mapreduce.Job:  map 0% reduce 0%
16/03/23 22:49:40 INFO mapreduce.Job:  map 50% reduce 0%
16/03/23 22:49:41 INFO mapreduce.Job:  map 100% reduce 0%
16/03/23 22:49:49 INFO mapreduce.Job:  map 100% reduce 50%
16/03/23 22:49:53 INFO mapreduce.Job:  map 100% reduce 100%
16/03/23 22:49:53 INFO mapreduce.Job: Job job_1455656680779_0161 completed
successfully
…
```

4. After that, you should have a "counts.txt" on your local folders that contains all words and their counts.

```
[NetID@login-1-1 mrlab]$ head counts.txt
"Project    5
#51302]     1
'AS-IS',    1
("the 1
($1   1
(1/12 1
(1/3  1
(10   1
(1066 1
(10_s._),   2
```

Please note that the above "run.sh" script takes 7 input arguments:

```
./run.sh mapper.py reducer.py book.txt output 2 2 counts.txt
```

./run.sh	
mapper.py	an executable mapper script (in this case, written in Python)
reducer.py	an executable reducer script (in this case, written in Python)
book.txt	the path to input data on HDFS
output	the path to output directory on HDFS
2	the number of mappers (aka suggesting the number of map tasks)
2	the number of reducers (aka specifying the number of reduce tasks)
counts.txt	the path to a local file for the merged output from all reducers

In the above example, since we ran with 2 reducers, there are two output "parts" on HDFS. We can verify the output from each reducers by listing the contents of the output folder on HDFS:

```
[NetID@login-1-1 mrlab]$ hadoop fs -ls output
Found 3 items
-rw-r--r--   3 hvo hvo          0 2016-03-23 22:49 output/_SUCCESS
-rw-r--r--   3 hvo hvo      53595 2016-03-23 22:49 output/part-00000
-rw-r--r--   3 hvo hvo      51174 2016-03-23 22:49 output/part-00001
```

Besides a status file indicating "_SUCCESS", there are also files starting with "part-". Each of these file is an output from a reducer. We specified 2 reducers, thus, we get 2 parts. "run.sh" also takes care of merging these parts together and write them to the last parameter "counts.txt". This last argument is optional, i.e. if we do not specify "counts.txt", there will be no merged output collected from HDFS to the client node.

If you'd like to learn more about the mechanics, please inspect the file "run.sh" further. This script file should work for the default settings of most Cloudera distribution of Hadoop. However, if you are working with a different Hadoop cluster, the only part that you'd need to change, for most of the time, in this file is the content of the variable STREAMING_JAR, pointing it to the hadoop-streaming.jar file of your Hadoop distribution.

The script is written for the class and is provided as "AS IS", please feel free to adapt it for your own usage.

TASK 4 – Chaining MapReduce jobs

In Hadoop Streaming, if we would like to apply multiple MapReduce phases to a data set (similar to what we did in previous homeworks and labs using the mapreduce.py package), we would have to launch multiple streaming jobs, i.e. a sequence of ./run.sh calls chaining output of the previous one to the input of the next one. For example, if our objective is to count the top 3 words of a book with the most occurences, we would normally run a top-k MapReduce algorithm on the output of the word count example. In our GIT repo, *mapper2.py* and *reducer2.py* should contain the code for performing the top-3 algorithm in Hadoop Streaming. An important note about the top-k algorithm in MapReduce is that it requires the number of reduce tasks to be exactly 1. This is because we need to have a global view of all top-k outputs from all mappers in order to compute it correctly.

To run this algorithm on our previous output, we can issue the following command:

```
[NetID@login-1-1 mrlab]$ ./run.sh mapper2.py reducer2.py output output2 2 1 top3.txt
rm: `output2': No such file or directory
16/03/23 23:41:13 INFO Configuration.deprecation: mapred.job.name is deprecated.
Instead, use mapreduce.job.name
16/03/23 23:41:13 INFO Configuration.deprecation: mapred.map.tasks is deprecated.
Instead, use mapreduce.job.maps
packageJobJar: [] [/opt/cloudera/parcels/CDH-5.4.5-1.cdh5.4.5.p0.7/jars/hadoop-
streaming-2.6.0-cdh5.4.5.jar] /tmp/streamjob8642491323405526165.jar tmpDir=null
16/03/23 23:41:14 INFO client.ConfiguredRMFailoverProxyProvider: Failing over to
rm356
16/03/23 23:41:15 INFO mapred.FileInputFormat: Total input paths to process : 2
16/03/23 23:41:15 INFO mapreduce.JobSubmitter: number of splits:2
16/03/23 23:41:15 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1455656680779_0163
16/03/23 23:41:15 INFO impl.YarnClientImpl: Submitted application
application_1455656680779_0163
16/03/23 23:41:15 INFO mapreduce.Job: The url to track the job:
http://hadoop11.cusp.nyu.edu:8088/proxy/application_1455656680779_0163/
16/03/23 23:41:15 INFO mapreduce.Job: Running job: job_1455656680779_0163
16/03/23 23:41:27 INFO mapreduce.Job: Job job_1455656680779_0163 running in uber mode
: false
16/03/23 23:41:27 INFO mapreduce.Job:  map 0% reduce 0%
16/03/23 23:41:38 INFO mapreduce.Job:  map 100% reduce 0%
16/03/23 23:41:45 INFO mapreduce.Job:  map 100% reduce 100%
16/03/23 23:41:45 INFO mapreduce.Job: Job job_1455656680779_0163 completed
successfully
…
```

After this, you should have a "top3.txt" file under your local folder listing the top 3 words:

```
[NetID@login-1-1 mrlab]$ cat top3.txt
the:2104
of:1631
and:1277
```

Two important observations here are: (1) the input path is "output", which is a folder specifying all files written from the previous job; and (2) the number of reducer is 1.

EXTRA 1 – How to test your Hadoop Streaming code offline

You can test your code mapper.py and reducer.py given an input file offline by using the test_run.sh script. The syntax is as follows:

```
[user@local ~]$ cat input.txt | ./test_run.sh mapper.py reducer.py
<your output will be printed to the console>
```

We can test our lab code by running the following:

```
[user@local mrlab]$ cat book.txt | ./test_run.sh mapper.py reducer.py | ./test_run.sh
mapper2.py reducer2.py
the:2104
of:1631
and:1277
```

EXTRA 2 – How to upload large/compressed file to HDFS

It is common that we have large, and often compressed files on our local machines, or even from a remote source that we would like to upload onto HDFS for further processing using Hadoop. Of course, we could first copy the data to **cluster.cusp.nyu.edu**, then use **"hadoop fs –put"** to upload them to HDFS. This method works well indeed for data sets that are already available in the CUSP Data Facility since those data should be mounted on the machine the same way you access them on **compute.cusp.nyu.edu**. However, if you have new data from outside of CUSP, this could be an issue, as writing the data to disk could unnecessarily take up storage space and processing time. This is even more complicated if we have a compressed file on local storage, but would like to uncompress them on HDFS. In these cases, a good approach is to use pipes (|) to stream external contents and uncompress them if necessary onto HDFS.

An example here is that we would like to analyze the CitiBike Monthly Trip Data, which are available as ZIP files at: https://s3.amazonaws.com/tripdata/index.html. In particular, let's assume we wanted to have the trips of February, 2015 to be uploaded onto HDFS as an uncompressed CSV file. A naive approach would be:

```
[hvo@cluster ~]$ wget https://s3.amazonaws.com/tripdata/201502-citibike-tripdata.zip
--2016-03-24 00:02:52--  https://s3.amazonaws.com/tripdata/201502-citibike-
tripdata.zip
Resolving s3.amazonaws.com... 54.231.8.192
Connecting to s3.amazonaws.com|54.231.8.192|:443... connected.
...

[hvo@cluster ~]$ unzip 201502-citibike-tripdata.zip
Archive:  201502-citibike-tripdata.zip
  inflating: 201502-citibike-tripdata.csv

[hvo@cluster ~]$ hadoop fs –put 201502-citibike-tripdata.csv .
```

As you can see, this approach requires us storing and uncompress the file to local storage on **cluster.cusp.nyu.edu**, where we would have to remove later on. A more "streaming" way that cuts the "middle-man" storage is to use pipe that fetch the data, uncompress and upload it directly to HDFS as follows:

```
[hvo@cluster ~]$ wget -O - https://s3.amazonaws.com/tripdata/201502-citibike-
tripdata.zip | gunzip - | hadoop fs -put - 201502-citibike-tripdata.csv
--2016-03-24 00:02:52--  https://s3.amazonaws.com/tripdata/201502-citibike-
tripdata.zip
Resolving s3.amazonaws.com... 54.231.8.192
Connecting to s3.amazonaws.com|54.231.8.192|:443... connected.
...
```

Again, two observations: (1) the dash symbol "**-**" is being used in place of filename to indicate the standard input; and (2) **gunzip** is being used in place of **unzip** since the former can handle data from the standard input while the latter cannot.

EXTRA 3 – How to monitor Hadoop jobs with Resource Manager

Modern Hadoop distributions come with YARN, which also includes a Web User Interface for monitoring and managing resources. This interface is usually hosted on the Name Node at port 8088. However, because these name nodes are usually behind a firewall, we cannot access them directly from our local web browser. Fortunately, at CUSP, we already expose this interface through the HUE platform. If you login into HUE portal at: https://data.cusp.nyu.edu, and select "Job Browser" on the upper right corner, you should be able to see all of your jobs being run.