Lab 3 – HDFS/Hadoop

CC5212-1 – March 31, 2021

Today we will play with HDFS (Hadoop Distributed File System) and Hadoop (Open Source implementation of a MapReduce framework). First, I will give you two example Hadoop jobs: one to count words and the other to sort the results by frequency; you must compile and run on the course's (very modest) cluster of four machines. Second, you will have to create your own job based on this example.

- First we want to SSH onto the master server of the class' cluster:
  - The machine we will use is accessible through SSH at `cm.dcc.uchile.cl:220`. To log in on a terminal (in Windows, Mac, Unix), you can run `ssh -p 220 uhadoop@cm.dcc.uchile.cl`.[1] I will give the password during the session.
    - If you are on Windows, head to `http://aidanhogan.com/teaching/tools/cc5212/` and download the tools found there. These are standalone (do not need to be installed) Windows tools for SSH and SFTP/SCP.
    - Open PuTTy. In host-name type `cm.dcc.uchile.cl` and port `220`. Click Save. Click Open.
  - Username and password will be provided during the class.
  - You are now on the master server of a ~~five~~ ☹ four-server cluster.
  - Since you are all using one username, please, please be considerate. **Please think twice before typing `rm -r`, `kill` and similar commands.** Please.[2]
- Next we want to look at the distributed file system (DFS) and upload some data to it; this file system is distributed over the four machines and is different from the local file system (LFS) of the master machine: for example, when you type `ls`, you see the files on the LFS of one master machine; when you type `hdfs dfs -ls`, you see the files on the DFS, which might be spread over thousands of machines! (or even four machines as in our case)
  - Type `hdfs dfsadmin -report` to see the state of the DFS.
  - Type `hdfs dfs`.[3] This shows you the options to interact with HDFS.
  - Type `hdfs dfs -ls /` to see the root contents. These files are stored across the servers.
  - Type `hdfs dfs -ls /uhadoop2021` to see that folder's contents. Then think up a user/group-name and afterwards type `hdfs dfs -mkdir /uhadoop2021/`USERNAME replacing USERNAME with your username, for example, `hdfs dfs -mkdir /uhadoop2021/ahogan`. This is now your personal folder on the **DFS**;[4] please keep your files for HDFS in this folder (e.g., output for later Hadoop tasks). Also beware that anyone can look at or even delete your data: do not upload your medical records or browser history to this folder.
  - On the local file system, go to directory `cd /data/uhadoop/shared/wiki`. Here you'll find the file(s) we want to do a word-count on stored locally. We're going to run a Hadoop job to count the words in this file. But how? Hadoop is designed to run on multiple machines while this file is on the LFS of the master machine ... but if we upload it to the DFS, all machines will have access to it and can process it through (e.g.) Hadoop. Note that there's a compressed version and an uncompressed version of the same file we worked with before. There is no difference between the files other than one is compressed and one is uncompressed. If you want to have a peek, type `cat es-wiki-abstracts.txt | more` or type `zcat es-wiki-abstracts.txt.gz | more`. Choose *one* file[5] and copy it to your folder in HDFS:
    `hdfs dfs -copyFromLocal es-wiki-abstracts.txt`.GZ ` /uhadoop2021/`USERNAME`/`
  - Check that it's there now: type `hdfs dfs -ls /uhadoop2021/`USERNAME. The data is now on the DFS. Actually you don't even know which of the four machines it's stored on and usually you don't care. But if you want more details, type `hdfs fsck /uhadoop2021/`USERNAME`/es-wiki-abstracts.txt`.GZ

[1] A useful application for Windows is PuTTy.
[2] Please.
[3] One can also type `hadoop fs` but it seems `hdfs fs` is preferred.
[4] Remember your directory on the **LFS** is `/data/2021/uhadoop/`USERNAME .
[5] Which would be faster for Hadoop to count words on? We will discuss in class!

- Now that we've gotten the data on the DFS, we need Hadoop code for counting words. Since this is your first foray into Hadoop, I've given you the code, which we will review in the class. You job is to compile and run it!
  - Download code from u-cursos and open it in Eclipse.
  - Open up the `CountWords` class. Here we see the mapper class, the reducer class and the main method. The map takes an input line and for each word `w`, outputs a pair `(w,1)`. The reducer then receives each word `w` as key and all the `1`'s (partial counts) associated with it and sums them, outputting for each word a pair `(w,s)` where `s` is the sum. We'll go through this code in the class.
  - Next we need to compile, package and run the word-count code on the cluster.
    * In the project in Eclipse, right click on `build.xml`, then `Run As ...`, then make sure `dist` is clicked and hit `Run`. If it fails, you may need to manually make a new `dist` folder in the project root. Refresh the project (F5) and make sure you have the JAR file.
      · If you get an error mentioning javac1.8 or similar, you need to right click on the `build.xml`, go to External Tools Configurations and add the line `-Dbuild.compiler=javac1.7` to arguments.
      · If you get an error saying something about `JAVA_HOME` not found, follow: `Window > Preferences >` Expand `Ant` tree `> Runtime > Highlight Global Entries > Add External Jars >` Then find and add `tools.jar` in the `lib/` folder of whatever Java JDK you have installed (e.g., on Windows, you might find it in `C:/Program Files (x86)/Java/jdk.../lib/tools.jar`).
    * We need to copy the JAR file to the server.
      · If on Windows, open WinSCP. Set `SFTP`. For the hostname, enter `cm.dcc.uchile.cl`. Enter `220` as port number and `uhadoop` as username. Don't enter the password yet. Click save. When it prompts, enter the same password as for PuTTy/SSH. Copy your `.jar` file into `/data/2021/uhadoop/USERNAME` folder.
      · If on Windows/Mac/Unix, in the terminal, enter:
      `scp -P 220 LOCAL-PATH/mdp-hadoop.jar uhadoop@cm.dcc.uchile.cl:/data/2021/uhadoop/USERNAME/`.
      Enter the same password as for SSH.
    * Now we just need to call the Hadoop job. Go back to PuTTy. Run (all one command):
      `hadoop jar /data/2021/uhadoop/USERNAME/mdp-hadoop.jar CountWords /uhadoop2021/USERNAME/es-wiki-abstracts.txt.GZ /uhadoop2021/USERNAME/wc/`.[6] Hopefully you will see the Map/Reduce progress as it happens.
    * When it's finished, look at the statistics and in particular the number of map and reduce tasks launched (this is the number of machines running the map and reduce for you). Now it's time to look at the results, which are stored on HDFS in the last argument of the previous command.
      · Run `hdfs dfs -ls /uhadoop2021/USERNAME/wc/` to see the output.
      · Run `hdfs dfs -cat /uhadoop2021/USERNAME/wc/part-r-00000 | more` to look into the file. Note that the words are ordered alphabetically so you might see some noisy characters at the top.
      · Try find the count for "`de`": run:
      `hdfs dfs -cat /uhadoop2021/USERNAME/wc/part-r-00000 | grep -P "^de\t" | more`
      Did you get 4,921,380?
  - These previous results are ordered alphabetically so we cannot easily find the most frequent words. Next revise and run the `SortWordCounts` code – a second Hadoop job to sort by frequency (descending) – on the previous results. The process is the same though the class and arguments change; run (all one command):
    `hadoop jar /data/2021/uhadoop/USERNAME/mdp-hadoop.jar SortWordCounts /uhadoop2021/USERNAME/wc/ /uhadoop2021/USERNAME/swc/`
    Note that we can provide the output folder of a Hadoop job as the input to the next job: it will consider all Hadoop files in the DFS folder as the input. Here we have manually chained two Hadoop jobs by hand: run one and then manually run the second on the output of the first. An obvious question then is: *how could we chain two jobs automatically?* (We'll discuss this in class.)
- Okay, that was a lot of instructions to see how to use HDFS and run Hadoop jobs, but we've still not practised coding anything for Hadoop. Hence your next task will be to count the stars.

[6] The output directory must *not* exist prior to running the job!

More specifically, we will count co-stars in movies that are listed in IMDb.[7] Your mission ... should you choose to accept it ... is to create a set of MapReduce jobs that count number of times pairs of actors or actresses have co-starred in the most movies together. Hopefully it shouldn't be impossible.[8]

- The data you need are on HDFS in the `/uhadoop/shared/imdb/` folder. The `imdb-stars.tsv` file is about 1GB and contains 16 million roles played by different actors in IMDb (pretty much the full database). Have a look inside the start of the file.[9] The header (not included in the file) is as follows:

Star Name	Movie Name	Year	Movie Number	Movie Type	Episode	Starring As	Role	Gender
...	...	...	...	...	...	...	...	...

  Columns are tab delimited. Some values may not apply and may be `null`.

  Star Name is the name of the star. The file *happens* to be sorted alphabetically so you might see stars with weird names in there at the start of the file (we do not need the file to be sorted for this task).

  Movie Name is the name of the movie or tv series, etc., that the star appears in.

  Year is ... well yes, the year the movie was released.

  Movie Number is used when a movie with the same name appears in the same year (e.g., `http://www.imdb.com/title/tt0801505/` is the second movie called "Crash" listed in 2004 so it will have `II` here). Often this will be `null` (if there was only one movie with that name in that year).

  Movie Type is the type of movie ... if it's a theatrical movie, a TV movie, a TV series, etc.[10]

  Episode is the name of an episode if it was a TV series.

  Starring As is the name of the actor/actress in the credits.

  Role is the character they played.

  Gender of the actor/actress.

  We will mainly focus on the first five columns. The Star Name is unique for a person. **However, for movies, Movie Name is not unique ... only the combination of Movie Name, Year and Movie Number are enough together to uniquely identify a movie.** To make a unique key for a movie, you'll need to concatenate the values of these three columns; for example, `Crash##2004##II` rather than just `Crash`. We only want to consider (full cinematic) movies: specifically you should only consider input lines where the Movie Type is equal to `THEATRICAL_MOVIE`; if it is something else, skip it. To look for Al: `hdfs dfs -cat /uhadoop/shared/imdb/imdb-stars.tsv | grep -e "^Pacino, Al" | more`, flick through some results with Spacebar, using Ctrl+C to end.

- The exercise now is to adapt the previous example and create a sequence of Hadoop jobs to count the number of movies that each pair of actors/actresses have co-starred in. The output should look like:

```
De Niro, Robert##Pacino, Al      x
Cage, Nicholas##Kidman, Nicole   y
...                              ...
```

  ... where x is the number of theatrical movies that both Robert De Niro and Al Pacino star in, etc. (`##` is just a delimiter; you can use a tab or anything unambiguous). Some tips:

  * You will have to create your own classes today but you can continue to work in the `mdp-hadoop` project and you can copy/follow the previous examples given for counting words.
  * Only include theatrical movies and make sure the movie key you use is unique (see above).
  * In a reduce, you can only read from the `Iterable` once. It is a stream produced by the underlying merge-sort. If you wish to read the values twice, you need to store them in a temporary list; you can assume that all values (actors) fit in memory for one key (movie).
  * If you want to store objects from the map/reduce calls, do not store the wrapper objects (e.g., store `String` not `Text`). For example, in the reduce, the objects returned for the values in the `Iterable` object are the same object with the data updated each time; if you store these values into a list, you will have multiple copies of the same object pointing to the last data item.

[7] The Internet Movie Database ... `http://imdb.com/`

[8] (sorry)

[9] `hdfs dfs -cat [file] | more` pressing spacebar to scan, Ctrl+C to end.

[10] In the code later, the options will appear in `org.mdp.imdb.ActorMovieParser.MovieRole.MovieType`

    - Think carefully about whether or not you can use a combiner in each task!
    - The most common error is not getting the types of keys and values right in the Java generics; these errors will not be caught by the compiler, so double-check them before you run!
    - `/uhadoop/shared/imdb/imdb-stars-100k.tsv` contains a subset of 100,000 roles from the input file (which contains 13 million). To save time, **you should use this sample for testing your code**. When it works for the smaller file, you can try run the process over the full file.
    - You will need multiple jobs. It's okay to run them manually one-by-one. Check the output of one before moving onto the next. The output of one task becomes the input of the next.[11]
    - When you're calling your JAR from the command line, the first argument is the name of the class that has the main method you need. The second argument should be the location of the input file/directory. The third argument should be the location of the output directory (it should not exist beforehand).
    - Do not copy the input data into your personal folder ... it's too big to make lots of copies. Just pass the input location `/uhadoop/shared/imdb/imdb-stars.tsv` (or the sample). But do write the output to your personal folder on HDFS (e.g., `/uhadoop2021/USERNAME/imdb/out1/`).
    - When running you might get an error
      `Error: org.apache.hadoop.mapreduce.task.reduce.Shuffle$ShuffleError: ....`
      `Caused by: java.lang.OutOfMemoryError: Java heap space`
      If this error happens, don't panic, Hadoop will retry and hopefully succeed next time.[12]
      Otherwise, if you get a different error, please go ahead and panic.
  - Once you have all counts for all co-stars, (create and) run a last Hadoop job to order the the co-star descending by count to find out which pairs have starred together the most times in theatrical movies!

- OPTIONAL: How could we avoid having to run each part separately? Try write the code so you can call one main method to get a sorted count of the co-stars by frequency directly from the input file.

- QUESTIONS: How is the scalability of these methods? Which part of the code (if any) would be most likely to cause problems if we were to consider a lot more input data?

- Submit to u-cursos a zip with:
  - A plain text file called `results.txt` including:
    - The count for any word beginning with '`s`' in Spanish Wikipedia (from `CountWords`).
    - The counts of the top ten words by frequency in Spanish Wikipedia (from `SortWordCounts`).
    - The counts of the top ten co-stars by frequency in IMDb.
    - The count of a pair of co-stars you're a huge fan of in IMDb.
  - The classes you created for the IMDb co-star counting tasks.

[11] Again note that you can use the folder name of the previous output as the input argument of the next job (useful if there are multiple parts). Hadoop will consider all the files named, e.g., `part-r-XXXXX` as input to the map of the next job.

[12] It started happening recently, it only happens intermittently and we're not sure how to solve it ... though it seems the ratio of memory used during shuffling needs to be decreased on Hadoop.