W205 Exercise **2**: Elements of a Streaming application

Juanjo Carin

December 5, 2015

Introduction

The architecture of the Storm project is as depicted in Figure 1 of the instructions. The difference (or enhancement) is that this project has 20 `parse-tweet` bolts, each receiving about 5% of the tweets coming from the `tweet` spout, and 10 `count` bolts, each receiving instances of the same word coming from the 20 other bolts (i.e., all 20 `parse-tweet` bolts can receive tweets containing the word *hello*, but only one of the `count` bolts will receive the partial counts of that word coming from those 20 bolts; part of this is mentioned in **Step 1**). All the `count` bolts write to the `Tcount` postgres database. The **topology** can be seen in `screenshots/Topology.PNG`.

Though most of the Python files did not require a lot of changes, I tried to introduce them to enhance the (applicability of the) results. For example (see **Step 3** for more details), I filtered common words (like pronouns) which are not surprising to be very frequent but do not add too much to the context, I increased the queue of the spout to be able to process more tweets, etc.

During the latest Labs I've using the required AMI instance (`ucbw205_complete_plus_postgres_PY2.7`), making new images of it as I installed new programs. Hopefully, the files in this repository will also run in that instance (but, for example, I didn't find Postgres in it).

> I found particularly useful the application `tmux` (which I installed by running `sudo yum intall yum`), which allows to have several windows (or panes) of the instance being run, and thus allowing to run the Storm project in one of them, checking the DB in another, editing files with `vi` in another, and so on.

File structure

I cloned the repository with the instructions and files (https://github.com/UC-Berkeley-I-School/w205-labs-exercises; just the folder corresponding to this exercise) and the one that already contained my solution to Exercise 1 in the 50 GB EBS volume of the instance, named `/data`, inside a folder named `/w205`. The former was cloned in a folder named `/ex2` (and hence, the full route is `/data/w205/ex2/exercise_2/`), and the latter in `/exercises` (since the name of my—private—repository is "W205fall2015" and I created a folder called `/exercise_2`, the whole route where this PDF file and the rest of files are is `/data/w205/exercises/W205fall2015/exercise_2`).

The folder `/screenshots` contains 2 additional screenshots of both tree structures (in my own solution I've omitted the logs, but still not every file is visible): `OPTIONAL-screenshot-TreeStructure_ex2.PNG` and `OPTIONAL-screenshot-TreeStructure_exercise2.PNG`.

All the code corresponding to the Storm project is, of course, inside the folder `/data/w205/exercises/W205fall2015/exercise_2/EX2Tweetwordcount/`: the files that I've modified, including names, are in the `/src` and `/topologies` directories.

Most of the tasks required to perform each step are included in the files `stepX.sh`, as explained below.

Step 1 – Environment and Tool Setup

The file `step1.sh` (which, like all the others, must be made executable by running `chmod +x step1.sh`):

- installs the `pyscopg2` library,
- clones the repository with the required files and instructions (actually, just the folder related to this Exercise, not anything else) in the directory mentioned above,
- creates a Storm project called `/EX2Tweetwordcount`, in the local repository where my solution is,
- copies the files in the folder `/tweetwordcount` (i.e., the template for the project) in the folder of this new Storm project, and
- renames the topology to `EX2tweetwordcount.clj`.

So almost everything in this step can be run with `step1.sh`: the exceptions are the cloning of my own repository (which is not required) and the editing of the topology (which is optional).

There is no need to change the topology because the file names of the spouts and bolts, and the class names inside those Python files are correct: they are the ones that appear in the bolts and spouts inside the `/src` folder that we have copied from the `/tweetwordcount`directory. The only (optional) changes that I made were to increase the number of parallel Python processes to 20 (in the case of `parse.py`, which acts as mapper) and 10 (in the case of `wordcount.py`, which acts as reducer), to parallelize more these tasks (and hence make them more efficient).

Step 2 – Twitter application setup

The file `step2.sh`:

- installs the tweepy library,
- (optionally downgrades the requests library to avoid a warning message,) and
- runs the sample application to check that we're able to connect to Twitter.

I.e., it performs steps 2.1 and 2.3. Before doing that, I logged in to Twitter, created a new application and the corresponding access keys, and included them in `Twittercredentials.py` (inside the `ex2` folder: i.e., in `/data/w205/ex2/exercise_2/`) as well as in the spout of our new project (`/data/w205/exercises/W205fall2015/exercise_2/EX2Tweetwordcount/src/spouts/tweets.py`). But these steps (2.2) are not required to run the project.

> **Posssible bug in `hello-stream-twitter.py`**
>
> I noticed a possible error in the code (though it's irrelevant to successfully complete the exercise): in line 31 tweets are lowered (`self.dataJsonText = self.dataJson["text"].lower()`)... but then in line 33 we look for `"Hello"` (a word with capital letters, so the condition will be never met), so only a count (the number of tweets in 60sec) is printed out. If we change `Hello` to `hello` in that line (33) we actually get those tweets containing that word.

Step 3 – Application deployment

This step involved modifying the existing code of the spout and bolts (in the `./src/` folder of the project) in order to perform the tasks required. As mentioned, how these tasks interact and are connected is described in the topology (`./topologies/EX2tweetwordcount.clj`). All this editing is outside the scope of the file `step3.sh` (explained later).

`Tweets` Spout

Apart from including my credentials in it, I modified the line:

```
self._queue = Queue.Queue(maxsize = 100)
```

increasing `maxsize` to `10000` (setting it to `0` would cause the queue size to be infinite). Otherwise, an `Empty Queue Exception` appears after a very few minutes, so we capture not too many tweets.

`Parse` Bolt

This bolt (remember that there are 20 of them working in parallel) tokenizes the stream, emitting the separate words that form the tweet to the next bolt, which aggregates their count. It works perfectly without modifications, but I tried to make more sense of the results we get, by introducing the following additions:

- I stripped punctuation between letters (`aword = aword.lower()`; this is the least important improvement, and only corrects a few cases).
- I lowered the words (by running `aword = aword.lower()`), so different instances of the same word (e.g., `hello`, 'Hello', and `HELLO`) are counted as equal. There are some cases in which the use of uppercase letters may involve a different meaning, but overall the advantage we gain is greater.
- I excluded **stop words** which do not add information about the content of the tweet. I installed the `nltk` library, but downloading a corpus (like `stopwords`) has to be made manually, so I copied the file with the stop words in English to the project directory. But that file might not be available if the project is run remotely, so I inserted the whole list in the code (only 128 words like `i`, `me`, `my`, `myself`...) and included as a 3rd condition for a word to be emitted that it's not in that list.

`Wordcount` Bolt

All the changes in the original file are related to the 2nd task of this bolt, which acts as reducer: it does not only print out the current aggregate counts but also inserts (or updates) the tuples `(word, count)` in a `Tweetwordcount` table inside a postgres database called `Tcount`.

All the changes in the original file are related to the 2nd task of this bolt, which acts as reducer: it does not only print out the current aggregate counts but also inserts (or updates) the tuples `(word, count)` in a `Tweetwordcount` table inside a postgres database called `Tcount`.

The code can be found is `/data/w205/exercises/W205fall2015/exercise_2/EX2Tweetwordcount/src/wordcount.py`. When we initialize the bolt, apart from initializing the counter (a dictionary) we open the database. During the process, the first step was to increment the count of the word (the number of times it has appeared); when this count is 1 (i.e., the 1st time), we insert the word (and that count of 1) in the table... if that word was not already in the table: previously the table `Tweetwordcount` is read and all the words put in a list; this is optional, but I included it to increase the table when the project is run more than once (otherwise an error occurs). If the count is greater than 1 (or the words already appeared in a previous execution of the Storm project), we do not insert the word but just update its count.

`step3.sh`

Before executing the project we need to:

1. Start postgres: using the script from a previous Lab `start_postgres.sh`.
2. Create the database: using a simple Python file (`createDB.py`) included in the root directory of this folder.

This is done by the script `step3.sh`, which also runs the Storm project. To debug my code I did not run simply `sparse run` but `sparse run 2>&1 | tee log.txt`, which writes the output and errors in a text file.

> For some reason I could not create a database called `Tcount`. I included the parameter `dbname=\"postgres\"`, which is the username, and that avoided the error... but the database was also called `postgres`.

I ran the project several times (as mentioned, the count of words that appeared previously gets updated without error), but for the next step (4) I initialized the database and let the project run for 20 minutes. The typical code I used to debug (after creating the database with `python createDB.py`) was as follows (or very similar):

```
sudo -u postgres psql


\dt
 SELECT word, count FROM tweetwordcount WHERE count>20;
DROP TABLE tweetwordcount;
\q
```

The Top 20 words (as well as the number of different words) can be found in the screenshot `screenshot-DEFINITIVE-Top20withoutStopWords-20min.PNG`. Compare these results with `screenshot-PREVIOUS-Top20withStopWords-Only10min.PNG`, which shows previous results when stop words are not filtered: in *half the time* there are 6,441 words, compared to 9,228, and the 20 most frequent words (almost all of them stop words) appear from roughly 100 to 600 times, while non-stop words appear from 120 to 380 times (in a period twice as long).

A screenshot of the Storm project running can be seen in `screenshots/screenshot-twitterStream-Final.PNG` (there's also another screenshot of the 1st version of the project, before writing results to postgres: `screenshot-twitterStream-BeforeSavingToDB.PNG`; it's basically the same because what we output to screen is the same).

Databases are stored in the directory `/data/pgsql/data/base/`. To know which one is the one we're interested in, we have to run (in postgres):

```
SELECT oid FROM pg_database WHERE datname = 'postgres';
```

I have included in a script neither that code nor the one needed to copy the database to our repository (`cp -r /data/pgsql/data/base/11564/ /data/w205/exercises/W205fall2015/exercise_2/DB`) , since the OID (`11564` in this case) will vary between executions.

Step 4 – Serving Scripts

The 2 scripts are located in the root folder of this repository. Here the script `step4_1.sh` just run some tests of `histogram.py` and `finalresults.py` with one argument (see `screenshot-finalresults_and_histogram.PNG` in the `screenshots` folder). The results of `final_results.py` were so long that it is run by a 2nd script, `step4_2.sh` (`screenshot-finalresults_withoutArguments-lastWords.PNG` shows words starting with `z` and `y`).

And there's yet another script, `step4_3.sh`, that (running a Python file called `top20.py`; both are in the root folder) plots the bar chart: I've tried to install `matplotlib` and other libraries in the instance but nothing works, so I've taken a "creative" approach. As requested, `Plot.PNG` is not in the `screenshots` folder but in the root.