Project Report - Management, Access, and Use of Big Data

Indiana University

Revision	Date	Description	Author
1.0	15-Oct-2016	Initial draft	Karthik Vegi
1.1	20-Oct-2016	Added description	Karthik Vegi
1.2	25-Oct-2016	Added visualization details, sources	Karthik Vegi

Project Description

In this project, a data pipeline is manually executed on a twitter dataset. The steps are applied on a subset of data, 10000 records to be precise.

Following are the steps along with the queries:

1) **Infrastructure Setup**
   - A new project is created on the Jetstream account
   - A new instance is added to the project by choosing an image
   - The instance is launched by setting the requirements for CPU and memory
   - The instance is launched
   - Once the instance is active, the commands are executed on the web shell
   - After a secure layer is a setup between the VM and the local machine using Putty, WinSCP is used to securely transfer files back and forth.
2) **Sourcing the Data**
   - The compressed twitter dataset is transferred to the VM by using WinSCP
   - The dataset is now un-compressed using tar command
     - *tar -zxf I590-TwitterProjectCode.tar.gz*
3) **Building the code**
   - The build file is modified to set up the project directory and java home
   - Go to the project folder and build the code by using the command
     - *ant*
   - Check the classes
4) **Data Pipeline - Reformat data**
   - The data format is not acceptable by MongoDB. We use a tool which is a shell script to change the data encoding to a tsv format using the below command:
     - *./bin/Reformat.sh users_10000.txt users_formatted.txt*
   - The following variable names are added as a header: *user_id user_name friend_count follower_count status_count favorite_count account_age user_location*
5) **Data Pipeline – Import data**
   - The data is now imported into MongoDB by using the following script:
     - ./bin/import_mongodb.sh projectA twitter tsv users_formatted.tsv
   - The parameters are database name, collection name, file type, and the filename.
6) **Data Pipeline – Query data**
   - In this step, we use the Google geocoding API which takes in the user location as input and updates the latitude and longitude for the user location.

- Since the geo-coding API allow processing only for 2500 records per day, the query is run over 4 days to update all the 10000 records.
- The following command is used to invoke the google geo-coding API
  - ./bin/QueryAndUpdate.sh ./config/config.properties projectA twitter ./input/query.json day1.log
- We can see that now the latitude and longitude have been added to the data


```
> use projectA
switched to db projectA
> db.twitter.find().pretty()
{
        "_id" : ObjectId("58095dcf678e5d137b2432d6"),
        "user_id" : 100008949,
        "user_name" : "esttrellitta",
        "friend_count" : 264,
        "follower_count" : 44,
        "status_count" : 6853,
        "favourite_count" : 0,
        "account_age" : "28 Dec 2009 18:01:42 GMT",
        "user_location" : "El Paso,Tx.",
        "geocode" : {
                "formatted_address" : "El Paso, TX, USA",
                "location" : {
                        "lat" : 31.7618778,
                        "lng" : -106.4850217
                }
        }
}
```


- Let us check the log statistics for all the days we processed the records:

/home/kvegi/projectA/I590-TwitterProjectCode/log

Name	Size	Changed	Rights	Owner
..		10/31/2016 12:21:34 AM	rwxr-xr-x	kvegi
test_geo_day4.log	7 KB	10/23/2016 2:12:44 PM	rw-rw-r--	kvegi
test_geo1.log	2,714 KB	10/20/2016 8:27:12 PM	rw-rw-r--	kvegi
test_geo2.log	2,902 KB	10/21/2016 11:43:29 AM	rw-rw-r--	kvegi
test_geo3.log	2,832 KB	10/22/2016 10:25:23 PM	rw-rw-r--	kvegi
test_geo4.log	51 KB	10/23/2016 1:56:27 PM	rw-rw-r--	kvegi
test1.log	1,277 KB	10/24/2016 11:10:32 PM	rw-rw-r--	kvegi

```
Total:10000 record(s) found.
Total:2777 record(s) processed.
Total:2774 record(s) updated.
```

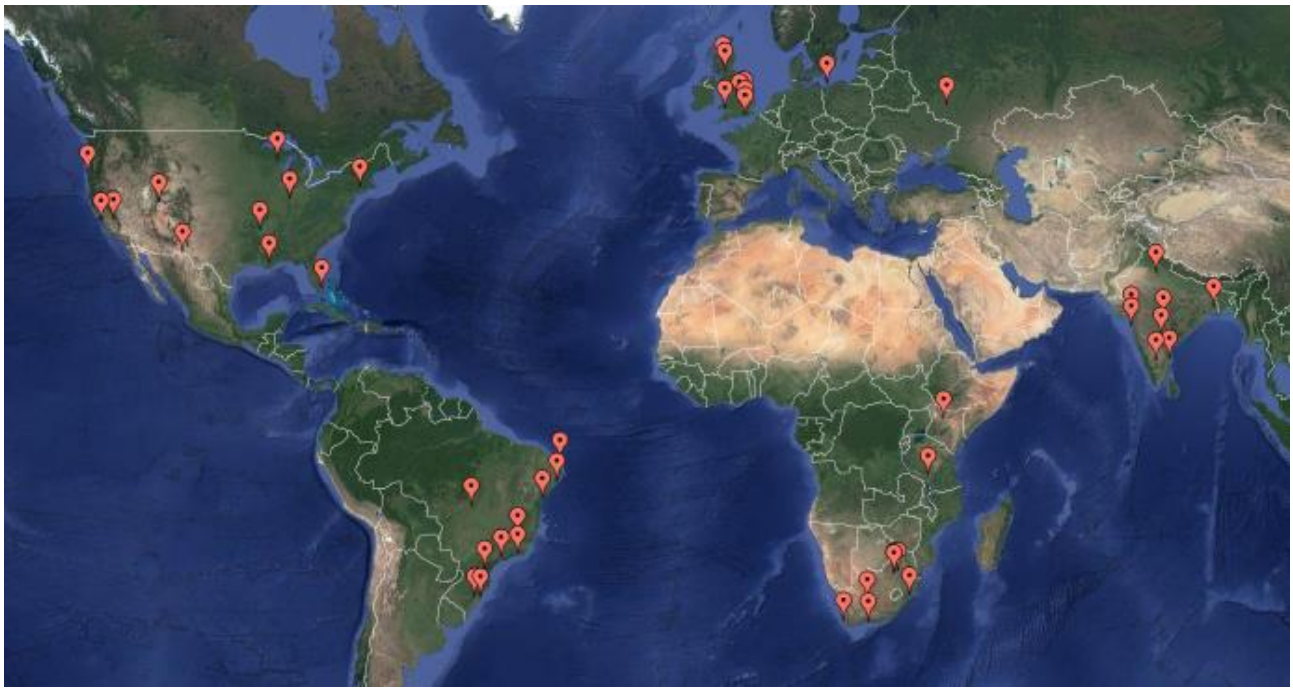
```
In this run:
Total:7226 record(s) found.
Total:2979 record(s) processed.
Total:2971 record(s) updated.
```

```
In this run:
Total:4255 record(s) found.
Total:2907 record(s) processed.
Total:2894 record(s) updated.
```

```
In this run:
Total:1317 record(s) found.
Total:1317 record(s) processed.
Total:1301 record(s) updated.
```

7) Data Pipeline – Visualization

- The next step is to select 50 profiles for visualization
- For the sake of randomness and completeness, the 50 records were filtered from USA, India, Brazil, Europe and Africa.
- Google map chart https://developers.google.com/chart/interactive/docs/gallery/map is used to plot the records on the map.
- The screenshot of the map is attached below:



(a) SOURCES:

- MongoDB manual: https://docs.mongodb.com/manual/
- Basics of shell scripting**:** http://www.shellscript.sh/
- VI Commands cheat sheet: http://www.lagmonster.org/docs/vi.html
- Info about Apache ant tool: https://en.wikipedia.org/wiki/Apache_Ant
- Jetstream user guide: http://www.jetstream-cloud.org/files/Jetstream_User_Guide-3-3-16-11am-Reduced.pdf
- Mongo import: https://docs.mongodb.com/manual/reference/program/mongoimport/
- Exporting data from MongoDB: https://docs.mongodb.com/v2.6/core/import-export/

- Google geocoding API: https://developers.google.com/maps/documentation/geocoding/intro
- Google Visualization map: https://developers.google.com/chart/interactive/docs/gallery/map

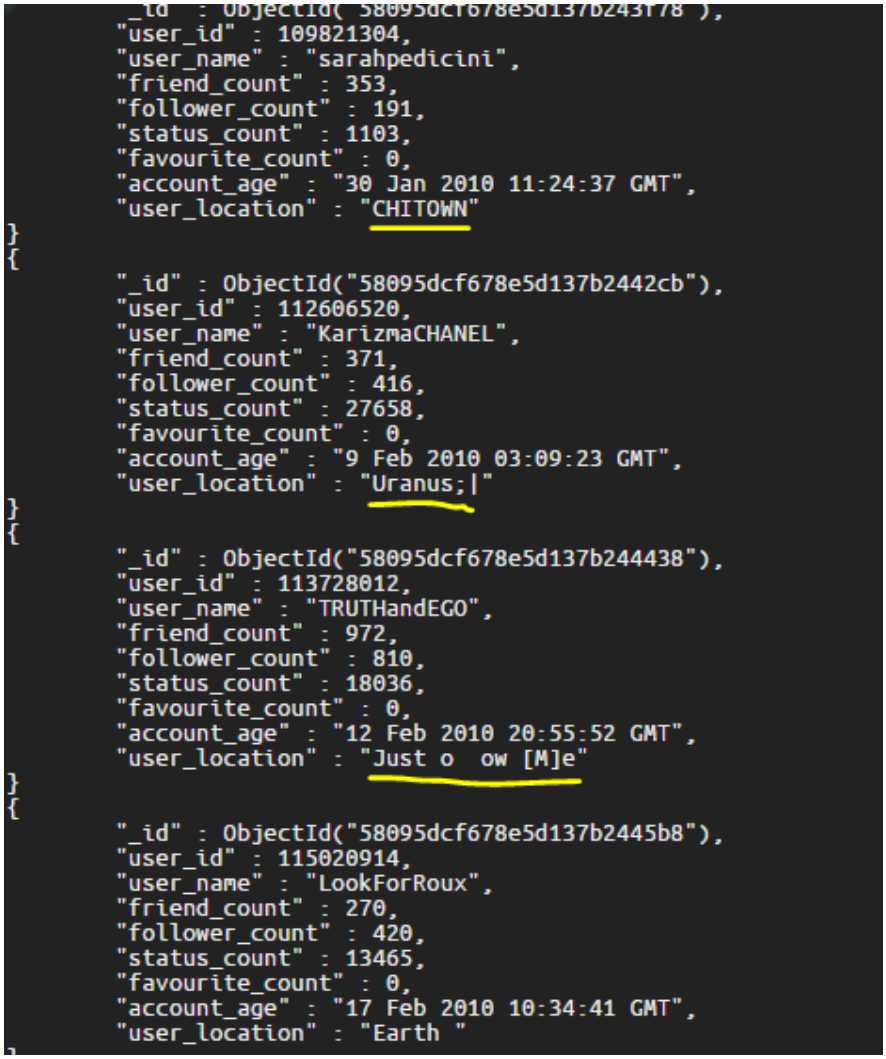
(b.i) How many locations were you able to validate?

- There can be 2 different cases here:

```
> db.twitter.find({geocode:{$exists: false}}).count()
16
> db.twitter.count({geocode: null})
2746
```

  - Geocode doesn't exist at all
    So, the geocode doesn't exist at all for 16 records.
  - Geocode exists and is null
    So, the geocode is null for 2746 records.

- Let us try to check those records



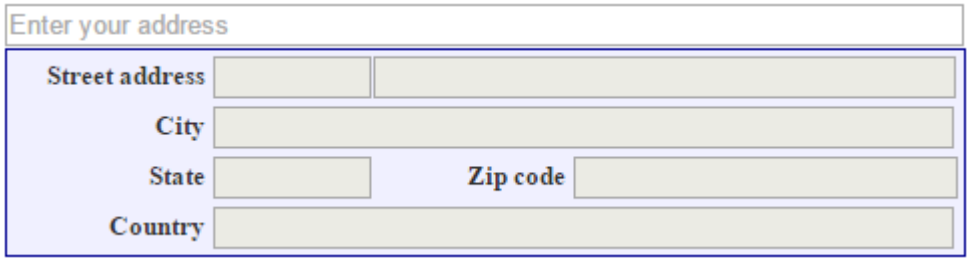
- As we can see that the reason is that the users have not set their location correctly.

How can resolve this issue?

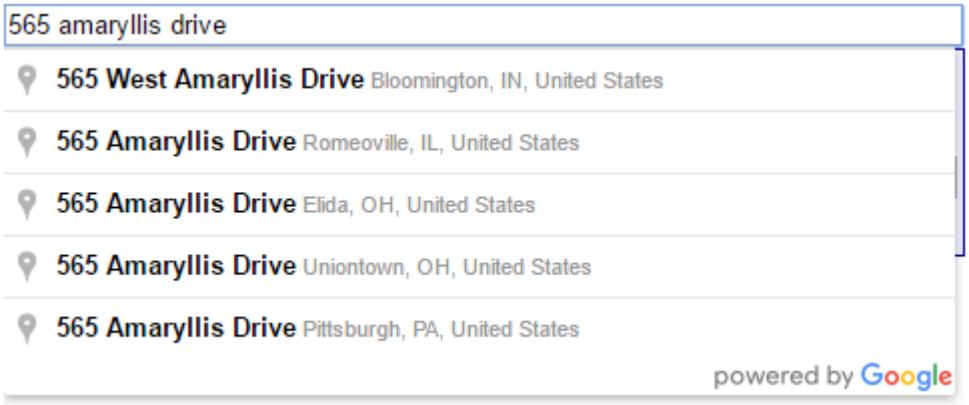
This issues is regarding the data quality. We can resolve the issue in two ways:

(i) Fixing the issue at the source(most-preferred)

- As we can see, this issue is because of the absence of validation procedures in the front-end system. The front-end system is not validating the location entered by the user and there is nothing we can do in the backend to correct these issues.
- It's important to note that when we have an issue with data, correcting the data at source is the best option. This way we can fix the current data as well as any future data coming from the source.
- The solution I would suggest is to validate the location when the user is entering it and prevent any locations which cannot be identified by the API. One useful API, in this case, is Google's autocomplete address form API which takes the user location, validates it and updates it correctly : https://developers.google.com/maps/documentation/javascript/examples/places-autocomplete-addressform



- Once you start typing your address it will auto pop-up the location by validating it.



- This will ensure that all the user locations are validated before the data percolates to the back-end data layers.

(ii) Fixing the data at back-end

Some of the data is close to correct meaning users have either entered a typo in the location. In this case, we could write a data transformation code which checks the locations and finds the closest match to a valid location. One way we could do this is by using cross data integration. We could use the name of the person to find his profile on other social networking sites like Facebook using the Facebook API and try to get the location: https://developers.facebook.com/docs/graph-api/reference/location/

This not only requires a lot of work; it means that we are applying a patch fix than fix the problem at the source. Hence the first solution is a preferred one.

(c) How can we improve the data pipeline?

The following are the steps that I think could improve the data pipeline:

- **Data collection**: Although we picked up the data directly in this project, we could have employed a batch job to source the data from twitter using the twitter API. This would be a great starting point. Working on the data starting from the source will help us identify the flaws in our front-end system and we can avoid discrepancies like the location.
- **Adding metadata**: Since the variables in our dataset are less, we couldn't see the benefit of adding metadata but usually real world data has more variables than observations in many cases which makes it difficult to understand the data. Metadata plays a crucial role in the data pipeline process.
- **Data Integration**: Since our data has some discrepancies like location, we could try to use Facebook's API to access the user location by the user name and we could fix some of the 2746 records for which the geo-location has not been populated.
- **Data Quality**: Data quality almost always follows integration. A quality check is done to ensure that none of the records are lost or modified in transformation and integration. The data is passed through quality check algorithms and a data quality report is sent to the data steward for a sign off.
- **Scheduler:** Since these steps have been done manually, we could try to automate them by using CRON jobs in Unix. For this to work, we need to create a shell script program which will pick up the pipeline jobs one at a time and runs it and generates a log. Whenever there is an error, it terminates and sends an email to the data analyst. The analyst will go

through the logs to find the issue, fixes it and restarts the flow. This could save a lot of time.

- **Tools that could be useful to improve the data pipeline:**
  - http://www.jamsscheduler.com/platforms/enterprise-job-scheduling/ for batch scheduling