Joel Cabrera and Venkata Krishnam Raju Datla

Data Management for Advanced Data Science Applications (16:954:694)

Professor John

May 2, 2021

Developing a Search Application of Twitter Data via MySQL and MongoDB

I. Introduction/Motivation

Our primary motivation was to understand the nuances of different Database technologies and to learn ways to store and retrieve data from these databases through the project. The reason for choosing tweets with hashtags related to Cryptocurrency is because it is an emerging topic in both academic and professional circles nowadays. Companies are also now investing in such currency due to its potential for high returns, resulting in more capital for said companies **[1]**. Thus, such a phenomenon makes one wonder how and why people talk about cryptocurrency in the first place, and what such discussion entails. Twitter is one of the most popular social hubs of people discussing topics, and cryptocurrency is no exception to this. Thus, it would be ideal that we pull Twitter data relating to cryptocurrency via the Twitter Streaming API, and process the data using database technologies.

II. Data & Description

Through the Twitter Streaming API, we were able to collect data of tweets on data science and subtopics/technologies related to it, such as deep learning, artificial intelligence, and Python. We pulled the data on April 12, 2021. we eventually scraped it for cryptocurrency data, as it was of further interest to us.

One should be aware of the fact that, as cryptocurrency is a constantly evolving subject, the collected data may be outdated over time. The number of tweets collected (for cryptocurrency) is exactly 11,000 and is rich enough to perform queries on. The hashtags used to collect and filter the data are as follows: **#doge, #dogecoin, #bitcoin, and #BTC**. We had decided to collect the data based on these hashtags because they denote some of the most popular subtopics relate to cryptocurrency, allowing for more data size and contents. The data collection was uninterrupted,

and the streaming was **directly written to a JSON file** by altering std.out . Below is how we saved and load the data (*Stream listener class is not displayed in the image*)

```
###Printed output directly to the JSON file
default_stdout = sys.stdout

sys.stdout = open('04.25.2020 tweets_5.json', 'w')
tweets_listener = StdOutListener()
stream = tweepy.Stream(api.auth, tweets_listener)
stream.filter(track=["#doge","#dogecoin","#bitcoin","#BTC"], languages=["en"])

sys.stdout = default_stdout
```

```
tweets_chk = open(r'04.25.2020 tweets_5.json',"r")
tweets_json = json.load(tweets_chk)
```

III. Databases & Search Application

Databases

Our Twitter data is very large and relatively complex (in terms of structure). As such, we would need to use database technologies to store, analyze, and process the data. Such database management systems (DBMSs) that are suitable for these tasks are MySQL **[3]** and MongoDB **[2]**. These DBMSs are relational and non-relational, respectively. We used these two DBMSs for storing and processing the data. The primary reason for choosing both these technologies is because of their widespread usage in numerous companies to store data. We believed that this would help us better understand these technologies and it will help **us with hands-on experience** which may help in our future careers. Both the installation and usage of these two databases for our project ended with success.

Both MySQL and MongoDB are free-to-use, open-source databases that facilitate effective management of databases. MySQL is globally renowned for being the most secure and reliable database management system used in popular web applications including WordPress, Drupal, Joomla, Facebook and Twitter. MongoDB is a document-based NoSQL database, a tool for storing JSON. MongoDB's document model allows virtually any kind of data structure to be modeled and manipulated easily. MongoDB's BSON data format, inspired by JSON, allows you to have objects in one collection have different sets of fields. 3,761 companies reportedly use MongoDB in their tech stacks, including Uber, Lyft, and Delivery Hero. In addition, both databases have an intuitive graphical user interface (GUI) to access the data (e.g. MySQL Workbench and MongoDB Compass, respectively). Such databases were ideal for loading, managing, and wrangling our data.

IV. Search application implementation – Storing data in Databases

We have stored and processed the data in MySQL and MongoDB through Project Jupyter, a free open-source integrated development environment (IDE) for Python using their corresponding Python libraries. We have used Pymongo to connect and interact with MongoDB databases and mysql-connector-python to connect to MySQL database. Some additional libraries were used for building the search application. These libraries are found below:

```
In [40]: import json
         import tweepy
         import sys
         import mysql.connector
         import pymongo
         import re
         import time
         from datetime import datetime
         import redis
         import fontstyle
         from IPython.display import clear_output
```

The central idea behind storing the data is for the user information to be stored in the MySQL database and tweets and retweets related information to be stored in MongoDB. We considered and discussed which attributes of the cryptocurrency JSON data based on what is shown in the search application. All the attributes that are stored are used in search application. We looked at the data dictionary for Twitter data **[5]** and considering the contents of certain attributes, we eventually decided to do the following: Into MySQL, we imported *Id_str, name, user (screenname), description (of tweet), followers, statuses, created_time, verified status, , friends_count, favorites_count, and protected status* into it. Into MongoDB, we imported *Id_str, text, time, user_id, user_handle, hash_tags_used (original tweet), and hash_tags_used (retweeted tweet) (and more) into it*. This storage execution can be summarized in the code figure below.

Before pushing attributes to DBs. We have performed few cleaning steps, for the user data that goes into MySQL, we removed quotations from description and user name fields and changed the format of the date field from tweet format to Python date format. We made sure that the user information is updated every time a new tweet is made by the existing user in the data base. In other words, the user DB reflects the attributes of the user from the most recent tweet.

The cleaning steps before inserting data into Mongo DB include taking tweet text and retweet text from extended tweet objects when the Truncated field of the tweet is set to True. This is also true for Hashtags, if the text is truncated, hashtags for the original and retweet are collected from the

extended tweet objects. A tweet is judged as a retweet if first 2 letters of the tweet is RT. We have also accounted for the improbable case where in the first two letters are RT but the tweet is not actually a retweet.

In terms of accomplishing the pushing of data into MySQL and MongoDB in Python, matching the schemata outlined previously, we do the following in Python below:

```
userdb = mysql.connector.connect(host="localhost",

if(userdb):
    print("Connection success")
else:
    print("Connection failed")

cursor = userdb.cursor(buffered = True)


cursor.execute("CREATE TABLE user (\
               user_id       CHAR(255),\
               user_name     CHAR(255),\
               handle        CHAR(255),\
               description   VARCHAR(500),\
              followers     BIGINT,\
               statuses      INT,\
               created_date  DATETIME,\
               verified      CHAR(5),\
               friends       INT,\
               favourites    INT,\
               protected     CHAR(50),\
               INDEX (user_name) )")
#cursor.execute('drop table user')
```

```
Connection success
```

```
client = pymongo.MongoClient("mongodb://localhost:27017/")
tweets_db = client["tweet_db"]
#tweets_col.drop()
tweets_col = tweets_db["tweets_col"]
```

Cleaning data and Pushing data into MySQL and MongoDB

```
for record in tweets_json:

    ###Starting with SQL insertion
    user = record['user']
    inp = [user['id_str'],user['name'], user['screen_name'],user['description'], user['followers_count'],
                user['statuses_count'], user['verified'],user['created_at'],user['friends_count'],user['favourites_count'],u

    #Cleaning commas from User name and Description
    inp[1] =  re.sub("'","",inp[1])
    if inp[3] is not None:
        inp[3] =  re.sub("'","",inp[3])

    #Converting tweepy created_date to datetime format
    inp[7] = datetime.strftime(datetime.strptime(inp[7],'%a %b %d %H:%M:%S +0000 %Y'), '%Y-%m-%d %H:%M:%S')


    #Updating the user information for the users who are already present in the user data base
    cursor.execute("select user_id from user where user_id ='{}'".format(inp[0]))

    result = cursor.fetchone()
    if result:
        query = "UPDATE user SET description = '{}',followers= {},statuses= {},verified= '{}',friends={},favourites ={},protect
    else:
        query = ("INSERT INTO user (user_id, user_name, handle,description,followers,statuses,verified,created_date,friends,fa
             "VALUES ('{}','{}','{}','{}',{},{},'{}','{}',{},{},'{}');".format(*inp))

    cursor.execute(query)


    #MongoDB insertion
    #Getting hashtags from the extended tweet if the original tweet is truncated
```

```
    entities =[]
    if record['truncated'] == False:
        text = record['text']
        for i in record['entities']['hashtags']:
            entities.append(i['text'])
    else:
        text = record['extended_tweet']['full_text']
        for i in record['extended_tweet']['entities']['hashtags']:
            entities.append(i['text'])

    try: #If the first 2 words of the tweet is RT but it is not actually retweet
        if record['text'][0:2] =='RT':
            retweet ="Y"
        #Getting hashtags and hashtags from the retweeted object
            rt_entities =[]
            if record['retweeted_status']['truncated'] == False:
                rt_text = record['retweeted_status']['text']

                for i in record['retweeted_status']['entities']['hashtags']:
                    rt_entities.append(i['text'])
            else:
                rt_text = record['retweeted_status']['extended_tweet']['full_text']

                for i in record['retweeted_status']['extended_tweet']['entities']['hashtags']:
                     rt_entities.append(i['text'])
        else:
            retweet ="N"
            rt_text = None
            rt_entities = None
    except:
        retweet ="N"
        rt_text = None
        rt_entities = None


    time = datetime.strftime(datetime.strptime(record['created_at'],'%a %b %d %H:%M:%S +0000 %Y'), '%Y-%m-%d %H:%M:%S')

    mydict = {"tweet_id":record['id_str'],"time":time,"user_id":record['user']['id_str'],"handle":record['user']['screen_name'],
              "followers":record['user']['followers_count'],"likes":record['favorite_count'],"hash_orig":entities,"hash_rtwt":rt
             "rtwt_text":rt_text }

    tweets_col.insert_one(mydict)

tweets_col.create_index("followers")
```

V. Search application implementation – Search functions and GUI

Given that we were able to successfully connect to the 2 databases and push certain JSON attributes into each database, we were then able to develop a search application for our cryptocurrency data. The application can be used to either search by hashtags, words, users, and time range (date). For hashtags, we have considered a particular tweet associated with a hashtag (e.g. #BTC) if the hashtag is present in either one of original hashtags or retweeted hashtags. We have implemented similar logic for word search, if the searched word is present in either of original tweet or text of a retweet, the tweet is considered to be associated with the word. For users, we queried for the number of tweets the user posted and their number of followers, friends, location, and date of account of creation. We have also pulled the most recent tweet posted by the user. For time range, we have filtered the data based on the selected time range. We defined these searches as four different functions and called the functions based on the inputs from the user. Then, we developed a GUI where the user inputs the item that they want to search for, and then receive the output created from the item's function. Below are snippets of 1 of these functions and the GUI in Python (*note that these functions also have connections to Redis **[4]**, which will be explained later*).

Creating functions for the search application

```
def hashtag(z):
    try:
        redis_search = "hashtag_"+z

        if conn.exists(redis_search):
            rec = conn.hgetall(redis_search)
            avg_follow=rec['avg_follow']
            dist_users=rec['dist_users']
            tweets_cnt=rec['tweets_cnt']
            tweet_1=rec['tweet_1']
            tweet_2=rec['tweet_2']

        else:
            myquery = {"$or":[{ "hash_orig": { "$elemMatch": {"$eq":z} } },{ "hash_rtwt": { "$elemMatch": {"$eq":z} } }]}
            mydoc = tweets_col.find(myquery).sort("followers",-1)

            tweets_cnt = tweets_col.count_documents(myquery)
            dist_users = len(tweets_col.distinct('user_id', myquery))

            #ave_query =([{"$group": {"_id":'null', "average": {"$avg":"$followers"} } }])

            p =1
            avg = 0
            for i in mydoc:
                avg = avg+i['followers']
                p+=1

            avg_follow = round(avg/p,0)
            mydoc = tweets_col.find(myquery).sort("followers",-1)

            tweet_1=mydoc[0]['text']
            tweet_2=mydoc[1]['text']

        text = fontstyle.apply('SUMMARY STATISTICS:', 'bold/white/black_BG')

        summary="""

        Number of tweets: {}

        Number of users who posted the hashtag: {}

        Average no. of followers for the users who tweeted: {}

        """.format(tweets_cnt,dist_users,avg_follow)


        twt_head = fontstyle.apply('Two Tweets from the most followed people ', 'bold/white/black_BG')

        tweet = """

        1.{}

        2.{}""".format(tweet_1,tweet_2)


        print(text,summary,twt_head,tweet)
    except:
        error = fontstyle.apply('Use alternate search criteria', 'bold/white/black_BG')
        print(error)
```

```
sig ="Y"
while sig=="Y":
    print("The attribute you want to search on: hashtag, user, date or word?")
    x =input()
    if x=='hashtag':
        clear_output()
        print("what is the hashtag?")
        has = input()
        clear_output()
        hashtag(has)
    if x=='date':
        clear_output()
        print("Enter the from date and to date seperated by a comma in the foramt Y-m-d H:M:S ex: 2021-
        minm, maxm = input().split(",")
        clear_output()
        time_range(minm,maxm)
    if x=='user':
        clear_output()
        print("what is the user handle?")
        q = input()
        clear_output()
        user(q)
    if x=='word':
        clear_output()
        print("what is the word?")
        p = input()
        clear_output()
        word(p)
    print("Do we want to restart the application (Y/N?)")
    sig = input()
    clear_output()
print("Adios!")
```

```
Adios!
```

It is also important to note that, the search application outputs two sets of information: The first is the summaries of the number of tweets associated with the selected item, number of users, number of verified users, number of mentions in retweets, and number of mentions in original tweets. For user selection, the summaries display the number of tweets a person posted and their number of followers, friends, location, text of tweet, and date of tweet creation. The second set shows 2 tweets posted by people with the highest followers for all the searches except for the user search. For the user search, we have queried the most recent tweet tweeted by the user. We have decided to use followers as our search criteria because we believed that the tweets made by people with more followers are more influential. Note that not all JSON attributes were used in the pushing of the data into the databases or called upon in the item functions (e.g. timestamp of first/last tweet, IDs of tweets, etc.). Our goal was to use the most relevant attributes for our search application and make findings based on them, both of which were able to accomplish.

Indexing

At the start of initializing connections to our MySQL and MongoDB databases, we created single-field indices on 2 JSON attributes: one on "user_name" in MySQL and one on "followers" in MongoDB. As we used follower count as one of the criteria for querying the tweets in the search application. We used the "user_name" column to pull the user information in the search application. So, index on user_name could accelerate the search. While more complicated indices could have been implemented, we were unable to do so because of the possibility of index prefixes. The code for our indices can be found at the end of the previously shown code.

VI. Caching using Redis & Search performance results

Redis

We used Redis to cache our data. Caching is important because it allows us to improve the querying time of our code. We have implemented caching for hashag and user searches. In terms of the procedure for caching our data, it can be summarized as follows: First, we queried the results of 4 popular hashtags (e.g. #dogecoin, #doge, #bitcoin, #btc) that the search application displays and stored these results as a dictionary against the hashtag key in Redis. We used Redis hash data structure to store the data. For user related caching, we followed a similar approach wherein we stored the queried results of 50 most popular users in Redis, the results of the user search includes

user_name, handle, description (of tweet), followers, statuses, created_date (of tweet), verified status, friends, favorites, and protected status. Then, we built our hashtag and user search functions that go into the search applications in such a way that the code searches for data in the Redis store first and queries the results if the hashtag/user is present in Redis. If not present, then the function proceeds to query the data from MongoDB/MySQL. A snippet of the Redis data storage code is shown below (*Redis search application implementation is present in the earlier code snippet*)

Instantiating Redis connection for caching

```
conn = redis.StrictRedis(host='127.0.0.1', port=6379,charset="utf-8", decode_responses=True)
```

Caching data for 4 most famous hashtags

```
hashtags = ["dogecoin", "doge", "bitcoin", "btc"]

for m in hashtags:
    myquery = {"$or":[{ "hash_orig": { "$elemMatch": {"$eq":m} } },{ "hash_rtwt": { "$elemMatch": {"$eq":m} } }]}
    mydoc = tweets_col.find(myquery).sort("followers",-1)

    tweets_cnt = tweets_col.count_documents(myquery)
    dist_users = len(tweets_col.distinct('user_id', myquery))

    #ave_query =([{"$group": {"_id":'null', "average": {"$avg":"$followers"} } }])

    p =1
    avg = 0
    for i in mydoc:
        avg = avg+i['followers']
        p+=1

    avg_follow = round(avg/p,0)
    mydoc = tweets_col.find(myquery).sort("followers",-1)

    tweet_1 = mydoc[0]['text']
    tweet_2 = mydoc[1]['text']

    mydict = {"tweets_cnt":tweets_cnt,"dist_users":dist_users,"avg_follow":avg_follow,"tweet_1":tweet_1,"tweet_2":tweet_2}

    key = "hashtag_"+m

    conn.hmset(key,mydict)
```

```
C:\Users\rajuk\anaconda3\envs\myenv\lib\site-packages\ipykernel_launcher.py:28: DeprecationWarning: Redis.hmset() is deprecate
d. Use Redis.hset() instead.
```

Caching data for 50 most popular users

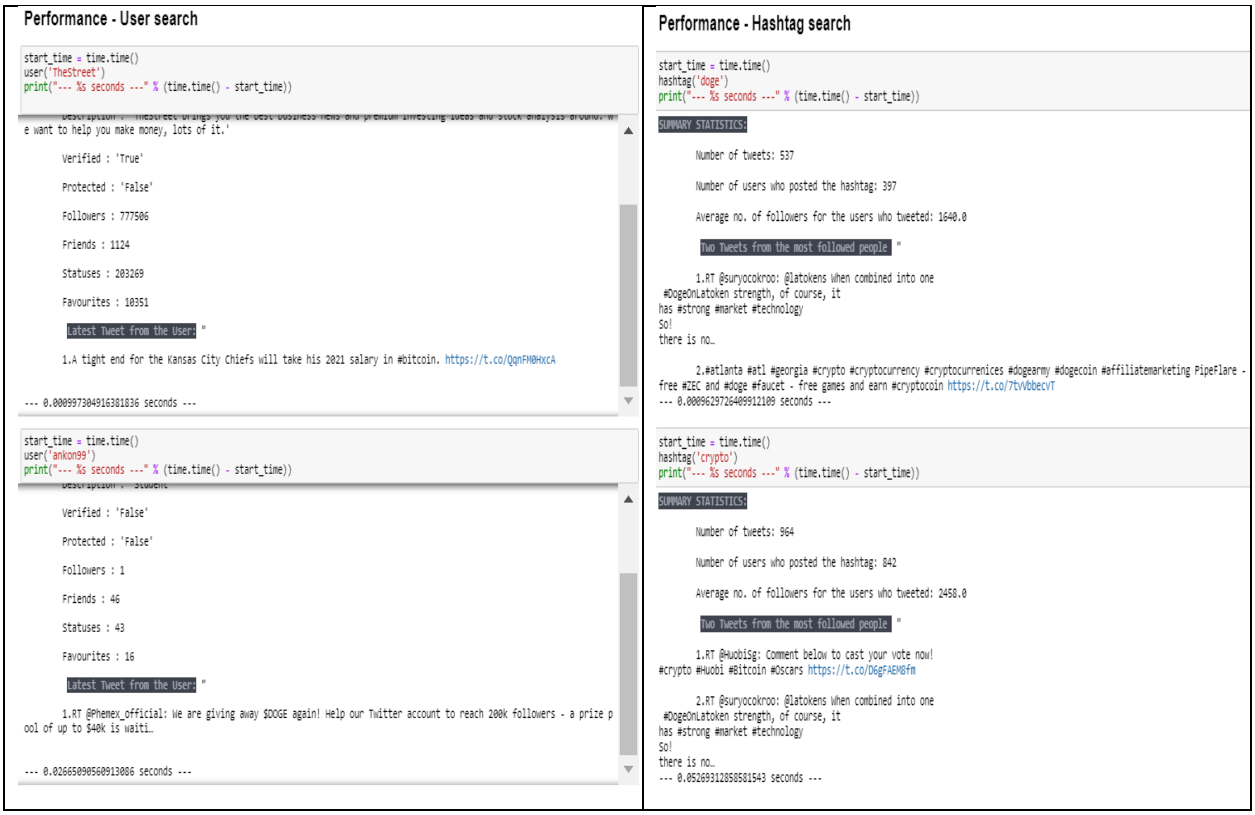
```
query ="select handle from (select handle from user order by followers desc) as b LIMIT 50"
cursor.execute(query)

x = cursor.fetchall()
top_50 = list(map(lambda x:x[0],x))

for i in top_50:

    query ="select user_name,handle,description,followers,statuses,created_date,verified,friends,favourites,protected from user
    cursor.execute(query)

    x = list(cursor.fetchone())
    x[5] = datetime.strftime(x[5], '%Y-%m-%d')

    myquery = { "handle": {"$eq":i}}
    mydoc = tweets_col.find(myquery).sort("time",-1)
```

Test Search Queries/Performances

Given the cached data for user and hashtag, we were now able to conduct timings of our test search queries and compared those of the cached data to those of the non-cached data.

For user search, the timings of the cached vs. non-cached data were approx. 0.000997 seconds and 0.026651 seconds, respectively. For hashtag search, the timings of the cached vs non-cached data were 0.000963 seconds and 0.052693 seconds, respectively.

Given the timings of the cached vs non-cached data for both items above, it can be said that the performance of searching for these 2 items is very faster when said items are cached in memory. The cached data is queried staggering 55 times faster than the non-cached data. The code and output for these test searches are found below.



VII. Work load distribution:

We have divided the presentation, coding, and report work almost equally. To be specific, in coding, while Joel took care of the tweets streaming and data storage, Raju took care of search application and indexing.

VIII. Conclusions/Learnings

Developing a search application for cryptocurrency based on data pulled from the Twitter Streaming API required careful planning, organization, and processing of its infrastructure. Having stored certain JSON attributes into MySQL and MongoDB and fully considered the search application design, we were then able to develop search functions for hashtag, user, word, and time range. Then, we cached some data for hashtag and user via Redis, and time its search queries. Comparing these timings to those of non-cached data search queried, we found that cached data loads much faster than non-cached data. In addition to better understanding the Database technologies we have learned important things about DB project execution: understand the data, carefully plan out the architecture of the application, become familiar with the database

technologies required for storing the data associated with this application, and apply techniques like caching judiciously to facilitate faster data access.

References

[1] Gavin Brown Associate Professor in Financial Technology. "Bitcoin: Why a Wave of Huge Companies like Tesla Rushing to Invest Could Derail the Stock Market." *The Conversation*, 28 Apr. 2021, theconversation.com/bitcoin-why-a-wave-of-huge-companies-like-tesla-rushing-to-invest-could-derail-the-stock-market-154966.

[2] "The Most Popular Database for Modern Apps." *MongoDB*, www.mongodb.com/.

[3] *MySQL*, www.mysql.com/.

[4] *Redis*, redis.io/.

[5] "Tweet Object | Twitter Developer." *Twitter*, Twitter, developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet.