EDS_1_twitter_big_data_nosql_datenanalyse

January 8, 2021

1 Schnittstellen von Twitter

Semesterarbeit 1

1.1 EDS-Einführung in Data Science

Klasse: BSc INF-P-IN010, BE1, HS20/21 Dozent: Dr. Tim vor der Brück Autor: Sandro Bürki, Michael Friderich Datum: 05.09.2020

1.2 Abstract / Executive Summary

In diesem ersten Teil der Semesterarbeit für das Modul EDS - Einführung in Data Science werden vier Fragen zum Nachrichtendienst Twitter beantwortet. Dabei stehen die Kommunikationsschnittstelle API, die Systemarchitektur und die Datenmengen im Vordergrund. Zuerst wird der Zugriff auf die API-Schnittstelle mittels Python Code vorgeführt. Dabei werden auch die Einschränkungen des Zugriffs auf die API-Schnittstelle aufgezeigt. Die Benutzeranfragen werden durch die Systemarchitektur abgearbeitet, wobei mehrere Load Balancer die Anfragen verteilen. Dabei geschieht entweder ein Fan-Out bei Benutzern mit wenigen Followern oder ein Speichern des Tweets bei Benutzern mit sehr vielen Followern. Um diese Datenmengen bewältigen zu können werden für die spezifischen Use-Cases relationale sowie NoSQL Datenbanken eingesetzt.

1.3 Einleitung

In diesem ersten Teil der Semesterarbeit im Modul EDS - Einführung in Data Science, werden die folgenden Fragen zur Kommunikationsschnittstelle API, der Systemarchitektur und der Datenmenge beantwortet. 1. Wie kann man auf einen Twitter Stream zugreifen - zeigen Sie, wie Sie Ihre eigenen Topics abrufen. 2. Welche Einschränkungen hat das Twitter API? 3. Wie werden die Anfragen in der Systemlandschaft von Twitter abgearbeiteten (nur auf Ebene Systemarchitektur) 4. Welchen Stellenwert haben NoSQL Datenbanken bei der Durchführung dieser Aufgabe?

1.4 Materialien und Methoden

Einige der Fragestellungen dieser Arbeit sind nicht direkt wissenschaftlicher Natur, weshalb sich die Methodik dieser Arbeit von einem wissenschaftlichen Artikel unterscheiden wird. Generell dienen das Lehrbuch Mining the Social Web (Russell/Klassen 2019), sowie die offizielle Twitter-Dokumentation (Twitter (o. J.): Twitter API v1.1) als primäre Informationsquellen.

EIn Twitter Stream ermöglicht den Zugriff auf Tweets in Echtzeit. Diese Streams können z. B. nach Stichwörtern, Sprache, und Ort gefiltert werden. Mithilfe des Codes in Listing 1 wird eine Suchabfrage abgesetzt, die uns aktuelle Tweets für das Stichwort “Open Source” anzeigt.

Um Zugriff auf die Twitter API zu erhalten, wird ein Developer Account benötigt. Mithilfe dessen lassen sich die Zugangsdaten für eine automatisierte, programmierbare Datenabfrage generieren. Diese Schlüssel können im oben aufgeführten Python Code an den angegebenen Platzhaltestellen eingesetzt werden. Nach dem erfolgreichen Login wird der/die gewünschte/n Suchbegriffe abgefragt, die aktuellen Tweets abgerufen und ausgegeben. Die Abfrage wird weitergeführt bis Sie manuell abgebrochen oder durch andere Verbindungsunterbrüche beendet wird. Hierbei müsste das Programm und somit die Connection erneut gestartet werden.

Um die Fragestellung bezüglich dem Stellenwert von NoSQL Datenbanken beantworten zu können, müssen die Anforderungen von Twitter mit den Unterschieden sowie den Vor- und Nachteilen von NoSQL gegenüber klassischen, relationalen Datenbanken verglichen werden.

[12]:
```
import twitter
import Twitter_credentials

def oauth_login():
    auth = twitter.oauth.OAuth(Twitter_credentials.OAUTH_TOKEN,
                               Twitter_credentials.OAUTH_TOKEN_SECRET,
                               Twitter_credentials.CONSUMER_KEY,
                               Twitter_credentials.CONSUMER_SECRET)

    twitter_api = twitter.Twitter(auth=auth)
    return twitter_api

twitter_api = oauth_login()
print(twitter_api)
```

```
<twitter.api.Twitter object at 0x7f6b814627c0>
```

[13]:
```
# Finding topics of interest by using the filtering capabilities it offers.
# Query terms

import sys

q = 'Open Source' # Comma-separated list of terms

print('Filtering the public timeline for track={0}'.format(q), file=sys.stderr)
sys.stderr.flush()
```

```
Filtering the public timeline for track=Open Source
```

[14]:
```
# Returns an instance of twitter.Twitter
twitter_api = oauth_login();
```

[15]:
```
# Reference the self.auth parameter
twitter_stream = twitter.TwitterStream(auth=twitter_api.auth);
```

[16]:
```
# See https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data
streamm = twitter_stream.statuses.filter(track=q)
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
/usr/local/lib/python3.8/dist-packages/twitter/stream.py in␣
↪handle_stream_response(req, uri, arg_data, block, timeout, heartbeat_timeout)
    210     try:
--> 211         handle = urllib_request.urlopen(req,)
    212     except urllib_error.HTTPError as e:

/usr/lib/python3.8/urllib/request.py in urlopen(url, data, timeout, cafile,␣
↪capath, cadefault, context)
    221         opener = _opener
--> 222     return opener.open(url, data, timeout)
    223

/usr/lib/python3.8/urllib/request.py in open(self, fullurl, data, timeout)
    530             meth = getattr(processor, meth_name)
--> 531             response = meth(req, response)
    532

/usr/lib/python3.8/urllib/request.py in http_response(self, request, response)
    639         if not (200 <= code < 300):
--> 640             response = self.parent.error(
    641                 'http', request, response, code, msg, hdrs)

/usr/lib/python3.8/urllib/request.py in error(self, proto, *args)
    568             args = (dict, 'default', 'http_error_default') + orig_args
--> 569             return self._call_chain(*args)
    570

/usr/lib/python3.8/urllib/request.py in _call_chain(self, chain, kind,␣
↪meth_name, *args)
    501             func = getattr(handler, meth_name)
--> 502             result = func(*args)
    503             if result is not None:

/usr/lib/python3.8/urllib/request.py in http_error_default(self, req, fp, code,
↪msg, hdrs)
    648     def http_error_default(self, req, fp, code, msg, hdrs):
--> 649         raise HTTPError(req.full_url, code, msg, hdrs, fp)
    650

HTTPError: HTTP Error 420: Enhance Your Calm

During handling of the above exception, another exception occurred:
```

```
TwitterHTTPError                          Traceback (most recent call last)
<ipython-input-16-18a37049afc1> in <module>
      1 # See https://developer.twitter.com/en/docs/tutorials/
 ↪consuming-streaming-data
----> 2 streamm = twitter_stream.statuses.filter(track=q)
      3
      4 print(streamm)
      5

/usr/local/lib/python3.8/dist-packages/twitter/api.py in __call__(self, **kwarg
    332             return self._handle_response_with_retry(req, uri, arg_data,
 ↪_timeout)
    333         else:
--> 334             return self._handle_response(req, uri, arg_data, _timeout)
    335
    336     def _handle_response(self, req, uri, arg_data, _timeout=None):

/usr/local/lib/python3.8/dist-packages/twitter/stream.py in␣
 ↪_handle_response(self, req, uri, arg_data, _timeout)
    284         class TwitterStreamCall(TwitterCall):
    285             def _handle_response(self, req, uri, arg_data,␣
 ↪_timeout=None):
--> 286                 return handle_stream_response(
    287                     req, uri, arg_data, block,
    288                     _timeout or timeout, heartbeat_timeout)

/usr/local/lib/python3.8/dist-packages/twitter/stream.py in␣
 ↪handle_stream_response(req, uri, arg_data, block, timeout, heartbeat_timeout)
    211         handle = urllib_request.urlopen(req,)
    212     except urllib_error.HTTPError as e:
--> 213         raise TwitterHTTPError(e, uri, 'json', arg_data)
    214     return iter(TwitterJSONIter(handle, uri, arg_data, block, timeout,␣
 ↪heartbeat_timeout))
    215

TwitterHTTPError: Twitter sent status 420 for URL: 1.1/statuses/filter.json␣
 ↪using parameters:␣
 ↪(oauth_consumer_key=hhff9myUkucabaPajxuveAGHL&oauth_nonce=132883811180418233 )&oauth_signat
 ↪0&track=Open%20Source&oauth_signature=PDklvEs61o1pQqs9Y39ro%2BN6Dko%3D)
details: Exceeded connection limit for user
```

[17]:
```
for tweet in twitter.stream:
    print(tweet['text'])
    sys.stdout.flush()
```

```
# Save to a database in a particular collection
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-54d9fbf35dda> in <module>
----> 1 for tweet in twitter.stream:
      2     print(tweet['text'])
      3     sys.stdout.flush()
      4
      5 # Save to a database in a particular collection

TypeError: 'module' object is not iterable
```

1.5 Ergebnisse / Findings / Resultate

Mittels dem oben vorgestellten Python Code können Tweets anhand von Stichwörter abgefragt werden. Dadurch erhalten wir live dazu passende Tweets. Wir können nun den Suchbegriff verändern und sehen wie häufig neue Tweets erstellt werden. Bei prominenten Stichwörtern wie z. B. Justin Bieber und Donald Trump erhalten wir sehr viel neue Tweets bei weniger häufig genutzten Stichwörtern wie z. B. Open Source ist die Zeitspanne zwischen zwei Tweets deutlich grösser.

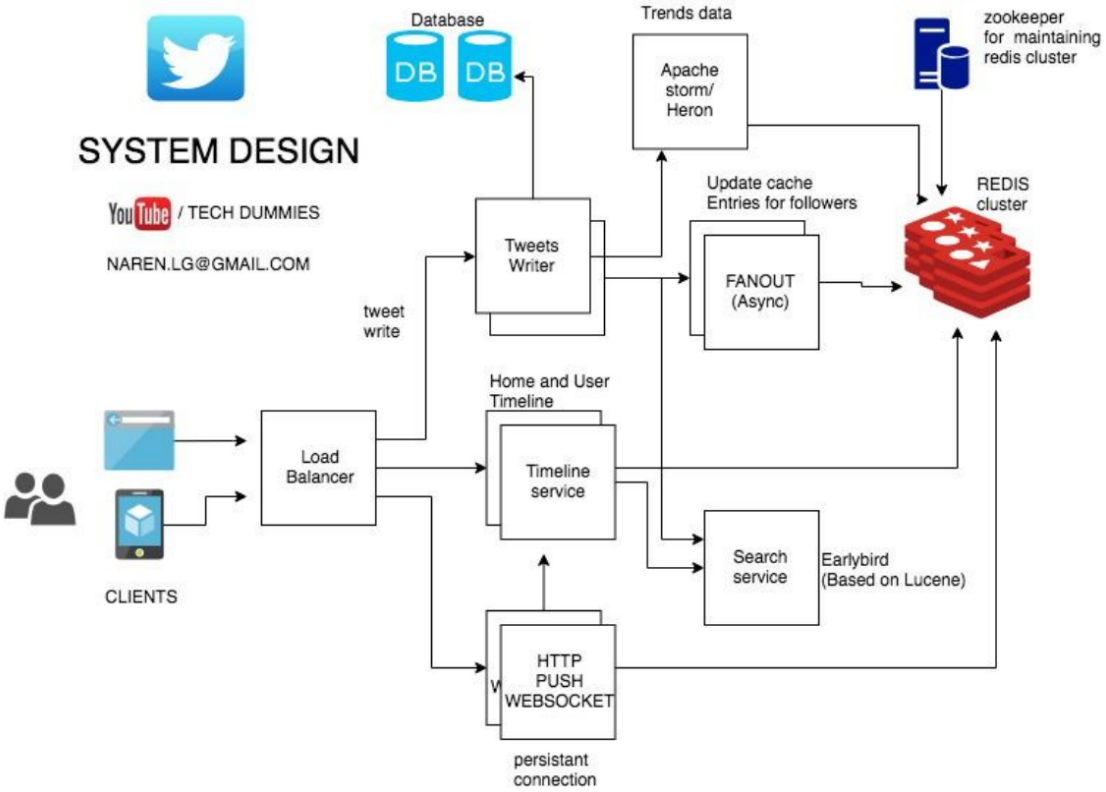
Bei jeglichen Abfragen gelten die Twitter Nutzungsbedingungen für Entwickler (Twitter (o. J.): Developer Agreement and Policy). Unter anderem umfassen diese Off-Twitter Matching, Spamming und legale Vorgaben. Für eine optimale Nutzung des Dienstes in einer kommerziell tauglichen Applikation müsste eine Fehlerbehandlung für die Ratenlimits (Twitter (o. J.): Rate limits) implementiert werden, sodass der Grund für ein Fehlverhalten klar ersichtlich wird. Die Limits sind nach dem spezifischen Endpoint und nach Art der HTTP-Requests, genauer den GET- und POST-Methoden gegliedert. Die verschiedenen Endpoints unterscheiden sich im Zeitfenster und in der Anzahl Anfragen, welche in dieser Zeit pro Applikation getätigt werden können. Für die GET-Endpoints beträgt ein Zeitfenster stets 15 Minuten. Für POST-Endpoints unterscheidet sich dies ferner. Als Beispiel ist nachfolgend der POST-Endpoint favorites/create abgebildet:

Endpoint	Rate limit window	Rate limit per user	Rate limit per app
POST favorites/create	24 hours	1000	1000

Für alle weiteren Endpoints wird auf die offizielle Twitter Entwicklerdokumentation (Twitter (o. J.): Rate limits) verwiesen, da eine komplette Auflistung hier den Rahmen sprengen würde. Der aktuelle Stand der Limits seine eigene Applikation betreffend lässt sich über einen GET-Endpoint (Twitter (o. J.): GET application/rate_limit_status) abfragen.

Eine Anfrage eines Client wird vom Load Balancer entgegengenommen. Es sind mehrere solcher Load Balancer vorhanden die sich in verschiedene Regionen aufteilen. Der Load Balancer sendet die Anfrage an ein REDIS Cluster. Dort wird nun ein sogenanntes Fan-out gemacht. Das bedeutet, die Anfrage wird in eine In-Memory Datenbank gespeichert und im Hintergrund werden alle betroffenen

Home Timelines angepasst. Gleichzeitig wird parallel der Tweet abgespeichert, damit dieser auch über die Suchfunktion von Nicht-Followern gefunden wird. Bei Benutzern mit mehreren Millionen Followern ist dieses vorgehen jedoch nicht geeignet da das Anpassen von so vielen Home Timelines zu viel Zeit beanspruchen würde. Hier wird die Home Timeline erst beim Aufruf durch den Benutzer aktualisiert.



Damit Twitters Architektur diese riesige Menge an Daten effizient persistieren, analysieren und verarbeiten kann, muss im Endeffekt die Datenbank den Anforderungen gerecht werden. Im Folgenden werden einige Themen angeschnitten, um die Viabilität von NoSQL-Datenbanken einzuschätzen.

1.5.1 Flexible Skalierbarkeit

Damit ein Unternehmen sich mit dem schnellen Lauf der Technologie agil weiterbewegen kann, ist Flexibilität von grosser Bedeutung. NoSQL-Datenbanken weisen diese Eigenschaft auf und sprechen somit dafür.

1.5.2 Grosse Datenmengen

Twitter verfügt bereits über eine riesige Menge an Daten, welche permanent abgefragt, verändert und erweitert wird. Klassische Datenbankmodelle wachsen zwar langsam an diese Anforderungen heran, dennoch ist NoSQL klar die präferierte Lösung.

1.5.3 Analysierbarkeit

Der primäre Nutzen von Twitter stellt die Analysierbarkeit der Daten dar. Dies ist jedoch nur bei einer passender Struktur möglich. Klassische Datenbanken erfüllen dieses Kriterium bereits seit

ihrer Entstehung, wobei NoSQL dies nicht zwingend erzwingt. Bei einem unüberlegten Datenmodell und stetiger Umgestaltung kann hierdurch ein Wildwuchs entstehen, welcher die Untersuchung beeinträchtigt. Aus diesem Grund ist es von hoher Bedeutung auch im NoSQL eine gut durchdachte Struktur der Daten zu wählen.

1.6 Fazit / Konklusion / Schlussfolgerung / Diskussion

Bei prominenten Stichwörtern sind die Abstände für neue Tweets sehr kurz. Dadurch lassen sich die riesigen Tweet-Mengen die Twitter pro Sekunde erhält erahnen und zu einem kleinen Teil sichtbar machen.

Wir sehen, dass Twitter verschiedene Herausforderungen meistern muss, um mit den riesigen Anfrage-Mengen umgehen zu können. Die Speicherkapazität steht dabei nicht im Vordergrund, da die einzelnen Textnachrichten lediglich 280 Zeichen umfassen können. Hingegen sind die Abhängigkeiten zwischen Benutzern und Tweets wichtig, denn gerade bei Benutzern mit sehr vielen Followern müssen im Hintergrund viele Abhängigkeiten zu anderen Timelines angepasst werden. Vermutlich könnte dies mit klassischen, relationalen SQL-Datenbanken schlechter bewerkstelligt werden, als mit NoSQL-Datenbanken. Für den NoSQL-Ansatz muss jedoch eine einheitliche Datenstruktur unter den Dokumenten etabliert werden, um die Analysierbarkeit zu gewährleisten. Den effektiven Einsatz von NoSQL hängt zudem vom jeweiligen Gebiet ab. So werden gemäss einer Präsentation (vgl. Weil, Kevin (2010): NoSQL at Twitter) von Kevin Weil mehrere Datenbankmodelle gleichzeitig geführt. Zudem bedeutet eine Umstrukturierung immer eine Investition von Ressourcen, was nicht unbedingt risikofrei und ökonomisch sinnvoll sein muss.

1.7 Gloassar

API Application Programming Interface ist eine Programmierschnittstelle um die Kommunikation zwischen Diensten zu erlauben.

Benutzer Bei Twitter können auch nicht reale Personen einen Account besitzen wie z.B.(Firmen, Imaginäre Personen, Bands)

Follower Man kann anderen Nutzern folgen (follow) und erhält dadurch Ihre Tweets.

NoSQL Eine Strukturform der Datenmodelle einer Datenbank in einer anderen Weise als die tabullare Charakteristik von klassischen, relationalen Datenbanken.

Timelines Die Home Timeline ist beinhaltet alle Tweets von Benutzern, denen man folgt. Die User TImeline , ist die eigene Benutzerspezifische Ansicht, die die eigens erstellten Tweets anzeigt.

Tweet Mit einem Tweet ist eine Kurznachricht gemeint, die maximal 140 Zeichen lang ist.

1.8 Quellenverzeichnis

1.8.1 Literaturverzeichnis

[1] L., Narendra (2018): System design for Twitter - Narendra L, in: Medium , [online] https://medium.com/@narengowda/system-design-for-twitter-e737284afc95 [05.09.2020].

[2] Russell, Matthew A. / Mikhail Klassen (2019): Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Instagram, GitHub, and More, 3. Aufl., Sebastopol, USA, California: O'Reilly Media.

[3] Success in Tech (2017): System Design: How to design Twitter? Interview question at Facebook, Google, Microsoft, in: YouTube , [online] https://www.youtube.com/watch?v=KmAyPUv9gOY [05.09.2020].

[4] Tech Dummies (2018): Twitter system design | twitter Software architecture | twitter interview questions, in: YouTube , [online] https://www.youtube.com/watch?v=wYk0xPP_P_8 [05.09.2020].

[5] Twitter (o. J.): Developer Agreement and Policy – Twitter Developers, in: Twitter Developer , [online] https://developer.twitter.com/en/developer-terms/agreement-and-policy [05.09.2020a].

[6] Twitter (o. J.): GET application/rate_limit_status, in: Docs | Twitter Developer , [online] https://developer.twitter.com/en/docs/twitter-api/v1/developer-utilities/rate-limit-status/api-reference/get-application-rate_limit_status [05.09.2020b].

[7] Twitter (o. J.): Rate limits, in: Docs | Twitter Developer , [online] https://developer.twitter.com/en/docs/twitter-api/v1/rate-limits [05.09.2020c].

[8] Twitter (o. J.): Twitter API v1.1, in: Docs | Twitter Developer, [online] https://developer.twitter.com/en/docs/twitter-api/v1 [05.09.2020d].

[9] Weil, Kevin (2010): NoSQL at Twitter, in: InfoQ , [online] https://www.infoq.com/presentations/NoSQL-at-Twitter/ [05.09.2020].