Project Report

PS Non-Standard Database Systems
Summer Term 2020

Department of Computer Sciences
University of Salzburg

Group Number 7

Zintl Ramona, 01322097

Kemperle Christoph, 01608238

Zivkovic Patrick, 01422129

June 17, 2020

Checkpoint 1: Planing Phase

1.1 Survey

In the following we are going to summarize four different papers on the topic of non-standard databases.

1. W. Vogels talks about eventually consistent databases. [11]: In many enterprises massive distributed database systems are necessary and indispensable. The CAP theorem states that only two of the shared-data-system properties (1) data consistency, (2) system availability and (3) tolerance to network partitioning can be achieved at the same time. Since network partitions must be a given, consistency and availability cannot be achieved simultaneously. Therefore, one of those properties need to be relaxed in some shape or form. By requesting a guarantee of high availability, the consistency needs to be weekend in order to ensure high performance standards of the system. When demanding a stronger consistency, the system may not be available at all times. The database engineer needs to reflect on the properties his system has to put emphasis on and has to apply those adequately. Most of the time a weaker consistency is implemented. This means that at times the access of an object will not the return the latest, updated value. However, after certain conditions are met the database will become eventually consistent again. The time frame, in which the database is inconsistent is dubbed the inconsistency window.
2. M. Stonebraker gives an overwiew on NewSQL opportunities [10]: Online Transaction Processing (OLTP) in combination with Extract-Transform-and-Load (ETL) were used in past with relational DBMS and data warehouses (old OLTP/SQL). Two customer requirements led to NewSQL: First, a much higher processing quantity of OLTP because of transaction requirements increased by Web applications which needs better DBMS performance and scalability. Second, the need for query capability of real-time analytics. This is not only neccessary in the case of Web applications but also for Non-Web applications of companies. Traditional OLTP with the Old SQL is not capable to deal with the huge workload and the

data warehouses are not current enough. Although NoSQL provides extreme scalability and high performance, not consideration of ACID and the avoidance of SQL leads to consistency problems and a huge programming workload. Hence the NewSQL seems to be the solution: high performance and scalability, ACID notion for transactions without application-level consistency and high-level language query capabilities of SQL.

3. A. Pavlo and M. Aslett discuss what's actually new with NewSQL [9]: NewSQL database systems are not a radical departure from existing system architectures but rather represent the next chapter in the continuous development of database technologies. Most of the techniques that these systems employ have existed in previous DBMSs. But many of them were only implemented one-at-a-time in a single system and never all together. What is therefore innovative about these NewSQL DBMSs is that they incorporate these ideas into single platforms. They are by-products of a new era where distributed computing resources are plentiful and affordable, but at the same time the demands of applications is much greater But they have had a relatively slow rate of adoption, especially compared to the developer-driven NoSQL uptake. This is because they are designed to support the transactional workloads that are mostly found in enterprise applications and are used to complement or replace existing RDBMS deployments, whereas NoSQL are being deployed in new application workloads.
4. Zaharia et al. talk about Spark - cluster computing with working sets[12]: They consider applications that reuse a working data set across multiple parallel operations. With MapReduce and similar programs, there is a performance loss due to iterative jobs and interactive analysis due to the repeated loading of data from the disk. In contrast, as a cluster computing framework, Spark supports applications with working sets while providing similar scalability and fault tolerance characteristics. That is because of resilient distributed datasets (RDDs) and the shared variables broadcast variables and accumulators. RDD means that a dataset can be reconstructed from the reliable storage and parallel operations can be done (*reduce, collect and foreach*). Spark is implemented in Scala which enables users to define RDDs, functions, classes and variables which can be used in parallel cluster operations. Examples are given which for example show how to make datasets persist accross several operations by creating cached RDDs.

1.2 System

We choose Apache Spark for our project.

Rationale Since we are a team consisting of three upcoming data scientist, we already stumbled upon Apache Spark in various job postings of renowned companies. While researching non-standard databases, this state-of-the-art technology of a general-purpose, fast data processing engine suitable for use in a wide range of data science applications quickly caught our eye. Being able to use interactive queries across large data sets, rapid processing of streaming data and most of all easy, accessible, integrated machine learning libraries further motivated us to get familiar with Apache Spark.
A comprehensive support for languages such as Java, Python, R and Scala in combination with our fondness of working in Python finally made up our minds to choose this system for our project.

In the following, we are going to discuss interesting features and properties of our chosen system.

1. **Lazy Evaluation & Speed:** Apache Spark holds intermediate results in memory rather than writing them to disk which is called RDD (Resilient Distributed Dataset) and is a read-only

collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. It is very useful especially when you need to work on the same dataset multiple times. It works both in-memory and on-disk. When data does not fit in memory Spark operators perform external operation. This can be used for processing datasets that are larger than the aggregate memory in a cluster. Spark will attempt to store as much as data in memory and then will spill to disk. It can store part of a data set in memory and the remaining data on the disk. With this in-memory data storage, Spark comes with performance advantage.

2. **Support of Multiple Languages & Frameworks:** In Spark, there is Support for multiple languages like Java, R, Scala, Python or SQL. Thus, it provides dynamicity and overcomes the limitation of other frameworks like Hadoop, that can build applications only in Java. Apache Spark also bundles libraries for applying machine learning and graph analysis techniques to data at scale. Spark MLlib includes a framework for creating machine learning pipelines, allowing for easy implementation of feature extraction, classification, regression, clustering, selections, and transformations on any structured dataset.
3. **Data Processing:** Spark allows to easily perform analysis in an SQL-like format over very large amount of data. Leveraging spark-core internals and abstraction over the underlying RDD, Spark provides what is known as DataFrames, an abstraction that integrates relational processing with Spark's functional programming API. This is done, by adding structural information to the data to give semi-structure or full structure to the data using schema with column names and with this, a dataset can be directly queried using the column names opening another level to data processing. In literal terms, Dataset API is an abstraction that gives an SQL feel and execution optimization to spark RDD by using the optimized SQL execution engine without also losing the functional operations that come with RDD.
4. **Real Time Streaming:** In comparison to traditional relational database systems Spark also provides real time streaming. While MapReduce is responsible for handling and processing already stored data Spark Streaming allows manipulation of real time data. This extension can easily manipulate streaming data as an abstraction over the underlying RDD in the form of Discretized Stream. Using the underlying RDD Spark core has two main advantages; it allows other core capabilities of Spark to be leveraged on Streaming Data as well as avail the data core operations that can be performed on RDDs. Discretized Stream of data means RDD data obtained in small real-time batches. Finally, processed data can be pushed out to filesystems, databases, and live dashboards.

1.3 Application Description

In a world in which data is generated extremely quickly, providing useful and meaningful results at the right time can provide helpful solutions not only for many areas that deal with data products, but also for a society as a whole. Since Social Media is one big player to collect people's insight and interests our goal is to analyze Twitter data. In particular, we want to extract the top hashtags (related to different topics during specific periods of time for example), to find out what the given discourse in a society is and therefore what people mostly discuss about. This not only helps companies or cities to find out what people are needing or wanting but also can help to see the spread of diseases, occurrence of natural disasters and many more.

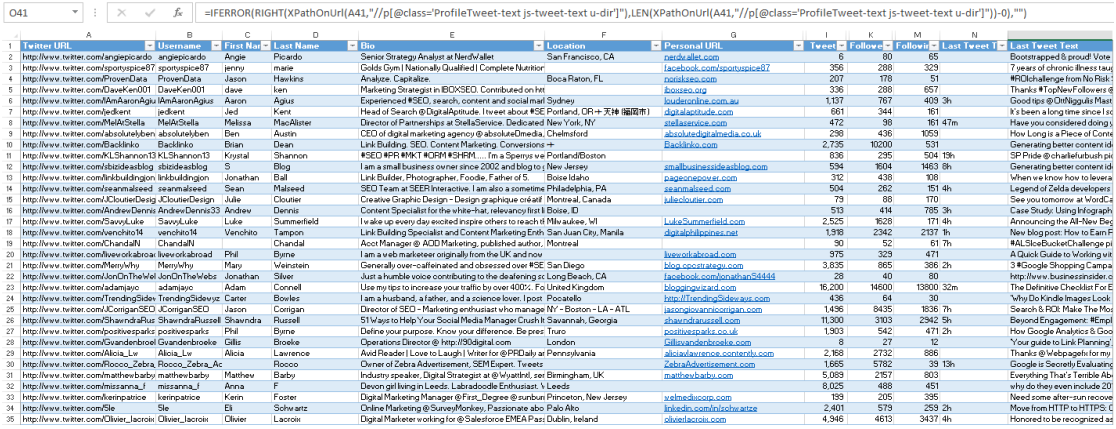
This application suits Apache Spark because it can handle big data in real-time and perform different analysis on it. Apache Spark also does fast big data processing employing Resilient Distributed Datasets (RDDs), streaming and Machine learning on a large scale at real-time.

Architectural Overview We are going to work with Python (pySpark) to utilize Spark Streaming component and use TCP Sockets to get connected to Twitter's streaming API through Tweepy library. After streaming tweets and having them on RDDs in Spark, we apply some

operations on tweets to extract the top hashtags. If possible, we would also like to filter the top hashtags according to various factors (region, time ...). Next, we save the top hashtags on a temporary SQL table using Spark SQL. Finally, we apply some visualization techniques in python to depict the results in a graph.

Experimental Data Because we want to analyze Twitter data our dataset consists of Twitter tweets (see below). Columns are:

1. Twitter URL
2. Username
3. First Name
4. Last Name
5. Bio
6. Location
7. Personal URL
8. Tweets
9. Follower
10. Following
11. Last Tweet Time
12. Last Tweet Text



Twitter URL	Username	First Nar	Last Name	Bio	Location	Personal URL	Tweet	Followe	Followi	Last Tweet T	Last Tweet Text
http://www.twitter.com/angiepicardo	angiepicardo	Angie	Picardo	Senior Strategy Analyst at NerdWallet	San Francisco, CA	nerdwallet.com	6	80	65		Bootstrapped & proud! Vote
http://www.twitter.com/sportyspice07	sportyspice07	jenny	marie	Golds Gym | Nationally Qualified | Complete Nutrition		facebook.com/sportyspice07	356	288	329		7 years of chronic illness tau
http://www.twitter.com/ProverData	ProverData	Jason	Hawkins	Analyze. Capitalize.	Boca Raton, FL	noisiseo.com	207	178	51		#HDIchallenge from No Risk
http://www.twitter.com/DaveKen001	DaveKen001	dave	ken	Marketing Strategist in IBOXSEO. Contributed on htt		iboxseo.org	336	288	657		Thanks #TopNewFollowers @
http://www.twitter.com/IAmAaronAgius	IAmAaronAgius	Aaron	Agius	Experienced #SEO, search, content and social marl	Sydney	louderonline.com.au	1,137	767	409	3h	Good tips @DtNigguls Mast
http://www.twitter.com/jedkent	jedkent	Jed	Kent	Head of Search @DigitalAptitude. I tweet about #SE	Portland, OR⇄天神 福岡市	digitalaptitude.com	661	344	161		It's been a long time since I sc
http://www.twitter.com/MeiAiStella	MeiAiStella	Melissa	MacAlister	Director of Partnerships at StellaService. Dedicated	New York, NY	stellaservice.com	472	98	161	47m	Have you considered doing y
http://www.twitter.com/absolutelyben	absolutelyben	Ben	Austin	CEO of digital marketing agency @ absoluteDmedia,	Chelmsford	absolutedigitalmedia.co.uk	239	436	1059		How Long is a Piece of Conte
http://www.twitter.com/Backlinko	Backlinko	Brian	Dean	Link Building. SEO. Content Marketing. Conversions +		Backlinko.com	2,735	10200	531		Generating better content id
http://www.twitter.com/KLShannon13	KLShannon13	Krystal	Shannon	#SEO #PR #MKT #ORM #SHRM..... I'm a Sperrys ve	Portland/Boston		836	295	504	13h	SP Pride @charliefurbush pi
http://www.twitter.com/sbizideasblog	sbizideasblog	S	Blog	I am a small business owner since 2002 and blog to ;	New Jersey	smallbusinessideasblog.com	594	1604	1463	8h	Generating better content id
http://www.twitter.com/linkbuildingjon	linkbuildingjon	Jonathan	Ball	Link Builder, Photographer, Foodie, Father of 5.	Boise Idaho	pagonepower.com	312	438	108		When we know how to lever a
http://www.twitter.com/seanmalseed	seanmalseed	Sean	Malseed	SEO Team at SEER Interactive. I am also a sometime	Philadelphia, PA	seanmalseed.com	504	262	151	4h	Legend of Zelda developers
http://www.twitter.com/JCloutierDesig	JCloutierDesign	Julie	Cloutier	Creative Graphic Design - Design graphique créatif	Montreal, Canada	juliecloutier.com	79	88	170		See you tomorrow at WordCa
http://www.twitter.com/AndrewDennis	AndrewDennis33	Andrew	Dennis	Content Specialist for the white-hat, relevancy first li	Boise, ID		513	414	785	3h	Case Study: Using Infograph
http://www.twitter.com/SavvyLuke	SavvyLuke	Luke	Summerfield	I wake up every day excited inspire others to reach th	Milwaukee, WI	LukeSummerfield.com	2,525	1628	171	4h	Announcing the All-New Beg
http://www.twitter.com/venchito14	venchito14	Venchito	Tampon	Link Building Specialist and Content Marketing Enth	San Juan City, Manila	digitalphilippines.net	1,918	2342	2137	1h	New blog post: How to Earn F
http://www.twitter.com/ChandalN	ChandalN		Chandal	Acct Manager @ AOD Marketing, published author,	Montreal		90	52	61	7h	#ALSIceBucketChallenge pi
http://www.twitter.com/liveworkabroad	liveworkabroad	Phil	Byrne	I am a web marketeer originally from the UK and now		liveworkabroad.com	975	329	471		A Quick Guide to Working wit
http://www.twitter.com/MemyWhy	MemyWhy	Mary	Weinstein	Generally over-caffeinated and obsessed over #SE	San Diego	blog.cpcstrategy.com	3,835	865	386	2h	3 #Google Shopping Campa
http://www.twitter.com/JonOnTheWeb	JonOnTheWebs	Jonathan	Silver	Just a humble voice contributing to the deafening sc	Long Beach, CA	facebook.com/jonathan54444	28	40	80		http://www.businessinsider.c
http://www.twitter.com/adamjayo	adamjayo	Adam	Connell	Use my tips to increase your traffic by over 400%. Fo	United Kingdom	bloggingwizard.com	16,200	14600	13800	32m	The Definitive Checklist For E
http://www.twitter.com/TrendingSides	TrendingSideviz	Carter	Bowles	I am a husband, a father, and a science lover. I post	Pocatello	http://TrendingSideways.com	436	64	30		'Why Do Kindle Images Look
http://www.twitter.com/JCorriganSEO	JCorriganSEO	Jason	Corrigan	Director of SEO - Marketing enthusiast who manage	NY - Boston - LA - ATL	jasongiovannicorrigan.com	1,496	8435	1836	7h	Search & ROI: Make The Mo
http://www.twitter.com/ShawndraRus	ShawndraRussell	Shawndra	Russell	51 Ways to Help Your Social Media Manager Crush It	Savannah, Georgia	shawndrarussell.com	11,300	3103	2942	5h	Beyond Engagement: #Empl
http://www.twitter.com/postnewsparks	postnewsparks	Phil	Byrne	Define your purpose. Know your difference. Be pres	Truro	postnewsparks.co.uk	1,903	542	471	2h	How Google Analytics & Goo
http://www.twitter.com/Gvandenbroek	Gvandenbroeke	Gilliz	Broeke	Operations Director @ http://90digital.com	London	Gillisvandenbroeke.com	8	27	12		'Your guide to Link Planning'
http://www.twitter.com/Alicia_Lw	Alicia_Lw	Alicia	Lawrence	Avid Reader | Love to Laugh | Writer for @PRDaily at	Pennsylvania	aliciajlawrence.contently.com	2,168	2732	886		Thanks @Webpagefx for my
http://www.twitter.com/Rocco_Zebra_	Rocco_Zebra_Ac	Rocco		Owner of Zebra Advertisement, SEM Expert. Tweets		ZebraAdvertisement.com	1,665	5782	39	13h	Google is Secretly Evaluating
http://www.twitter.com/matthewbarby	matthewbarby	Matthew	Barby	Industry speaker, Digital Strategist at @wyattnl, ser	Birmingham, UK	matthewbarby.com	5,089	2157	803		Everything That's Terrible Ab
http://www.twitter.com/missanna_l	missanna_l	Anna	F	Devon girl living in Leeds. Labradoodle Enthusiast. \	Leeds		8,025	488	451		why do they even include 20
http://www.twitter.com/kerinpatrice	kerinpatrice	Kerin	Foster	Digital Marketing Manager @First_Degree @sunburi	Princeton, New Jersey	wolfmediacorp.com	199	205	395		Need some alter-sun recove
http://www.twitter.com/Sle	Sle	Eli	Schwartz	Online Marketing @SurveyMonkey. Passionate abo	Palo Alto	linkedin.com/in/schwartze	2,401	579	259	2h	Move from HTTP to HTTPS: C
http://www.twitter.com/Olivier_Iacroix	Olivier_Iacroix	Olivier	Lacroix	Digital Marketer working for @Salesforce EMEA Pass	Dublin, Ireland	olivierlacroix.com	4,946	4613	3437	4h	Honored to be recognized as

Mostly relevant for us is "Last Tweet Text" which contains the hashtags and the topics and "Last Tweet Time" which contains how many hours or minutes ago the tweet was written. Moreover "Location" could be important for a filter variable but since it is not specified in every row this could be a problem.

1.4 Roadmap

Optional. Split your project into individual steps and provide a first roadmap for the semester. You may use the `vroadmap` environment (see `report.tex` for definition):

Checkpoint 2: Implementation

2.1 Resources

To obtain our data sets by Twitter, a deeper research on the tweepy library was required. With the help of the official documentation [1] we could understand the process of connecting to Twitter API and streaming from it.

2020-04-27	First meeting: allocation of tasks
2020-05-05	Reading literature
2020-05-10	Writing summary of chosen papers
2020-05-12	We choose Apache Spark; Features of Spark and reading more literature on Apache Spark
2020-05-12	Writing justification of system choice and highlighting interesting system properties
2020-05-13	Think about application scenario and find data sets
2020-05-17	Define application scenario and describe Twitter data set
2020-05-20	Go through all points of checkpoint one again regarding coherence
2020-05-22	Sign up to twitter
2020-05-27	Consult online resources about twitter data extracting
2020-05-30	Get Apache Spark running on the PC
2020-06-05	Manage to stream Twitter data via Apache Spark
2020-06-08	Visualizing top hashtags for various topics
2020-06-05	Setup and implementation description
2020-06-15	Complete checkpoint two
2020-06-24	Prepare presentation

To get a deeper insight to Apache Spark, we have been combing several sources. One of the first of the was a relatively simple and good-readable book [2]. It was useful with installing and setting up Apache Spark, but also to get a good overview about the Spark streaming framework and for example get a first feeling for the handling with dstream objects.

One major task for us was the dealing with the dstream object containing the RDDs. There we used several websites from the official Spark documentation. First, as we used python as our programming language we had to look up in the PySpark documentation [4], PySpark is the collaboration of Apache Spark and Python. Also very useful was the Apache Spark documentation [5]. There all functions are translated between scala, java and python.

As we were not the first to implement twitter streaming with spark, there were a few examples of code which helped us to build up our application [7, 8, 6].

For detailed questions or arising errors we searched online forums, mainly stack overflow.

2.2 Setup

The application was implemented on Windows 10 OS. The following programs were used during the whole setup, import and implementation process:

1. **Apache Spark 2.4.6 Pre-Built for Aapche Hadoop 2.7**: Apache Spark was used as our database system

2. **Apache Hadoop 2.7.2**: was used for implementation of Apache Spark
3. **Java 1.8.0**: was also used for implementation of Apache Spark
4. **Python 3.7.3**
5. **Anaconda 2019.3**: was used to run Jupyter Notebook

In order that everything works properly, we had to set **environment Variables** in Windows 10 [3]:

1. HADOOP_HOME (hadoop folder-path)
2. JAVA_HOME (java folder-path)
3. SPARK_HOME (spark folder-path)
4. PYTHON_HOME (python folder-path)
5. Set Python and Spark folder-path also in the Path variable of the environment Variables

Jupyter Notebook
With Jupyter Notebook 1.0.0 and the ipython 7.4.0 interpreter for python we implemented the application. Packages we used are:

1. findspark==1.3.0
2. matplotlib==3.0.3
3. pandas==0.24.2
4. py4j==0.10.7
5. pyspark==2.4.5
6. seaborn==0.9.0
7. textblob==0.15.3
8. tweepy==3.8.0
9. twitter==1.18.0
10. numpy==1.16.2
11. PySocks==1.6.8
12. jsonschema==3.0.1

2.3 Datasets

2.3.1 Import

For our application we use stream-based data sets of the very popular social networking platform Twitter on which people post messages called tweets, each of which can contain up to 140 characters. To be able to stream Twitter data a Twitter API is needed which can be obtained by registering a new app on developer.twitter.com. To register this app a lot of questions regarding the purpose must be answered. With the four authentication keys we get from there and with the help of the tweepy library the Twitter API can be accessed.

```
consumer_key    = '...'
consumer_secret = '...'
access_token    = '...'
access_secret   = '...'
```

Every tweet comes with a lot of variables concerning e.g. the user who wrote it, the location etc. (see infographic of variables). As we do a hashtag and sentiment analysis and in order to be efficient we only are interested in the text of the tweets. At this point, we do a little pre-

processing of the text of our tweets. Twitter offers the possibility to reply directly to specific tweets. If you do so, the text of the first tweet will also appear in the new one. Since we do not want to distort our later analysis because of double hashtags, we delete the text of the first tweet. Also for the sentiment analysis we are only interested in the new tweet, because it can differ substantially in the polarity of the sentiment. For example, the original tweet may have negative sentiment, but the new tweet that we are interested in can be evaluated positively. Therefore, the tweets are cleaned up before the data is sent to the spark application.

```
def on_data(self, data):
    try:
        msg = json.loads(data)
        if ('retweeted_status' in msg):
            if ('extended_tweet' in msg['retweeted_status']):
                print(msg['retweeted_status']['extended_tweet']['full_text'])
                self.client_socket.send((str(msg['retweeted_status']['extended_tweet']['full_text']) +
                                         "\n").encode('utf-8'))
        elif ('extended_status' in msg):
            print(msg['extended_status']['full_text'])
            self.client_socket.send((str(msg['extended_status']['full_text']) + "\n").encode('utf-8'))
        else:
            print(msg['text'])
            self.client_socket.send((str(msg['text']) + "\n").encode('utf-8'))
    except BaseException as e:
        print("Error on_data: %s" % str(e))

    return True
```

To receive tweets and send it to our spark application we also need to set up a socket. This is also the time when we filter for a specific topic (e.g. ”corona”, ”blacklivesmatter”) we are interested in. So only tweets which contain this word will be processed. There is also the possibility to choose the language, for example ”English”, which we have done, because it makes no sense for us to work on languages we do not understand. By binding the host / local machine address and a specific port which must be the same in the spark part of our application we can receive and send the tweets in a continuous data stream.

```
def send_tweets(c_socket):
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_secret)

    twitter_stream = Stream(auth, TweetsListener(c_socket), tweet_mode="extended_tweet")
    twitter_stream.filter(track=['blacklivesmatter'], languages = ["en"])
```

Therefore the data sets we use are generated in real-time and keep on growing as the number of tweets increases. The size of the data sets depend on various factors. If you choose a more popular topic like ”corona”, far more tweets will be streamed than for an unpopular topic. The time of day and whether the application runs on weekdays or weekends will also have a big impact on the number of tweets.

2.4 Implementation

2.4.1 Short Description

We are using pySpark to set up a Spark Streaming environment, which is used to handle real-time Twitter data. The tweets are obtained through a TCP socket over an in-place Twitter API

connection. A user can interact with this process by choosing keywords, by which the data flow of tweets is filtered. The texts of the incoming tweets are then saved in a DStream (discretized stream) object consisting of Resilient Distributed Datasets (RDDs) in the Spark environment. Next we apply a couple of transformation to the RDDs and finally save them into a temporary SQL table for later queries. One SQL table is comprised of the different hashtags found in the texts and their respective number of appearances during the stream. This data is used to visualize the most relevant hashtags of the topic in question. With the help of Spark Streaming the graph gets updated with additionally gathered data every minute. The other SQL table is only used to obtain the full tweets in order to apply a sentiment analysis on the individual texts. The different tweets are then categorized in either ”positive”, ”neutral” or ”negative” sentiment classes. These are then counted and plotted to provide an overall sense of sentiment regarding the discussion of the desired topic.

2.4.2 Detailed Description

To start, we import the package ”findspark” to pinpoint the folder in which our Apache Spark setup is located at. We also import the necessary packages SparkContext, StreamingContext and SQLContext from pyspark, which we are going to use for our streaming and data storing purposes.

```
#Importing relevant packages
import findspark
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SQLContext

#initialising spark directory
findspark.init('Spark directory on your system')
```

Next, we are initializing a SparkContext, which serves as the core for any spark applicational functionality. We are also setting up a StreamingContext with the batch interval parameter of 10. This means that the input stream will be partitioned into batches every 10 seconds. An SQL-Context is also set in motion, which will help us to store the obtained streaming data effectively.

```
#create new spark session
sc = SparkContext()
#initiating the StreamingContext with 10 second batch interval
ssc = StreamingContext(sc, 10)
#next we initiate our sqlcontext
sqlContext = SQLContext(sc)
```

In the next step we direct the live-strem input data obtained from our Tweepy module into the SparkContext using a socket. This socket object has to be configured with the same local IP address and port like in the Tweepy module to ensure a successful data migration. The input data is then saved into the ”lines” variable that will be updated with new data every 60 seconds using the window() function.

```
#initiate streaming text from a TCP (socket) source:
socket_stream = ssc.socketTextStream("127.0.0.1", 5555)
#lines of tweets with socket stream window of size 60 (60 seconds of time)
lines = socket_stream.window(60)
```

Now we define transformations onto this ”lines” variable, which stores a discretized Stream (dStream) object consisting of Resilient Distributed Datasets (RDDs). These transformations

are applied directly onto every single RDD inside our constantly updated dStream object. We are using lambda expression because they are fast and require less memory. First, we create a tuple ”Tweet” which we will use to save our transformed data in. The flatmap function splits the incoming text into words, of which the filter function only selects those starting with a hashtag symbol. The subsequent map function converts the words into lower cases and assigns a 1 to every word representing a hashtag. With the reduceByKey function all the numbers listed next to same words are aggregated into the count of that particular hashtag. This tuple comprised of the hashtag-word and its count is then saved into the previously created named tuple ”Tweet”. Next, we take this data from every available RDD in the same manner and convert it into a dataframe to save it into a temporary SQL table. For the purpose of just visualizing the most important hashtags we take only the five most prevalent ones and sort them in a descending order.
A very similar, but simpler approach is used to extract only the text of the tweets for the purpose of performing a sentiment analysis on those later on.

```
#create tuple for hashtag and counting and a named tuple Tweet
hashtagfield = ("hashtag", "count")
Tweet = namedtuple('Tweet', hashtagfield)

### Transformation on the incoming streaming data ###
#Splits to a list
text = lines.flatMap(lambda text: text.split(" "))
#Filter hashtag calls
hashtag = text.filter(lambda word: word.lower().startswith("#"))
#Lower cases the word
lowerWord = hashtag.map(lambda word: (word.lower(), 1 ))
#sum up the count of equal hashtags
getCount = lowerWord.reduceByKey(lambda a, b: a + b)
#stores the tuple in named tuple Tweet
store = getCount.map(lambda rec: Tweet(rec[0], rec[1]))
#Sorts RDDs by hashtag count, registers only top 5 hashtags and saves them to a temporary sql table
store.foreachRDD(lambda rdd: rdd.toDF().sort(desc("count")).limit(5).registerTempTable("tweets"))
```

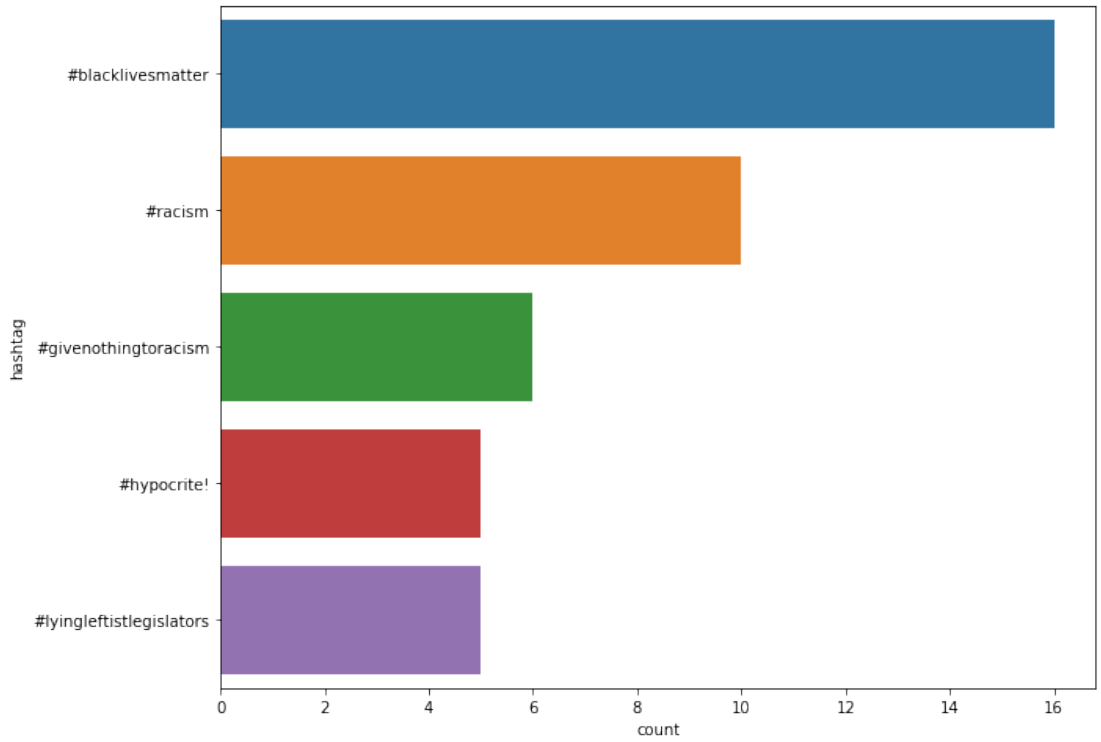
Now that the configuration is set up we are able to start the the stream. The aforementioned Tweepy module will now get initialized with the help of a simple shell, that opens said module, sets the user-defined search word and runs it. After executing this code chunk a command line window will open up that displays all the incoming tweets of the chosen topic.

```
#get user input for search
while True:
    search = input("Enter searchword: ")
    if(search != ""):
        print("Search for: " + search)
        break
    print("You have to use a searchword!")

search = search.split(",") #splits user input by ","

f = open("var.py", "w") #opens another file to store search variable
f.write("search = %s" % search) #overwrite the file
f.close()

#opens job.sh in shell command, which then calls the TweetsReceiver
file = subprocess.Popen("job.sh", shell = True)

ssc.start() #starts streaming
```

Next, we need to wait a minute to obtain enough tweets for a reasonable data analysis. Subsequently we can query the previously mentioned temporary SQL table to receive all the data that is already available up to this point. After converting it into a pandas dataframe, we are finally able to visualize the top 5 most predominant hashtags of the desired topic. This plot will

update itself automatically as it is given more data. In this example, we searched for the tweets containing the keywords ”racism” and ”police brutality”.

```
### Top 5 hashtags ###

#queries the temporary sql table and stores the result in a variable
top_5_tags = sqlContext.sql('Select hashtag, count from tweets')
#converts SQL variable to a pandas dataFrame
top_5_df = top_5_tags.toPandas()

#Visualization
plt.figure( figsize = (10, 8))
sns.barplot( x="count", y="hashtag", data=top_5_df)
plt.show()
```



To assess the current sentiment in the public discussion about a given topic, we perform a sentiment analysis on each tweeted text. For this purpose we use the function textblob from the package of the same name. This function returns two values: On one hand, the polarity of the text, a real-valued number between -1 and 1, that indicates how positive/negative the text appears. On the other, the subjectivity of the text, also a real-valued number between 0 and 1 assessing how subjective as opposed to objective the text appears. To obtain the text we first query our SQL table containing all the texts of the individual tweets.

```
from textblob import TextBlob #package for simple sentiment analysis

#gets temporary sql table and stores it in file
senti = sqlContext.sql('Select text from sentiment')
#convert file to pandas dataFrame
sen = senti.toPandas()
```

Since a tweet can contain various words that are not related to the content (e.g. https-weblinks), we first need to apply a cleaning function on it. We are aware that there are still more variables to account for which make the perfect cleaning of a text a non-trivial task. For instance are

abbreviations widely used in tweets but hard to detect.

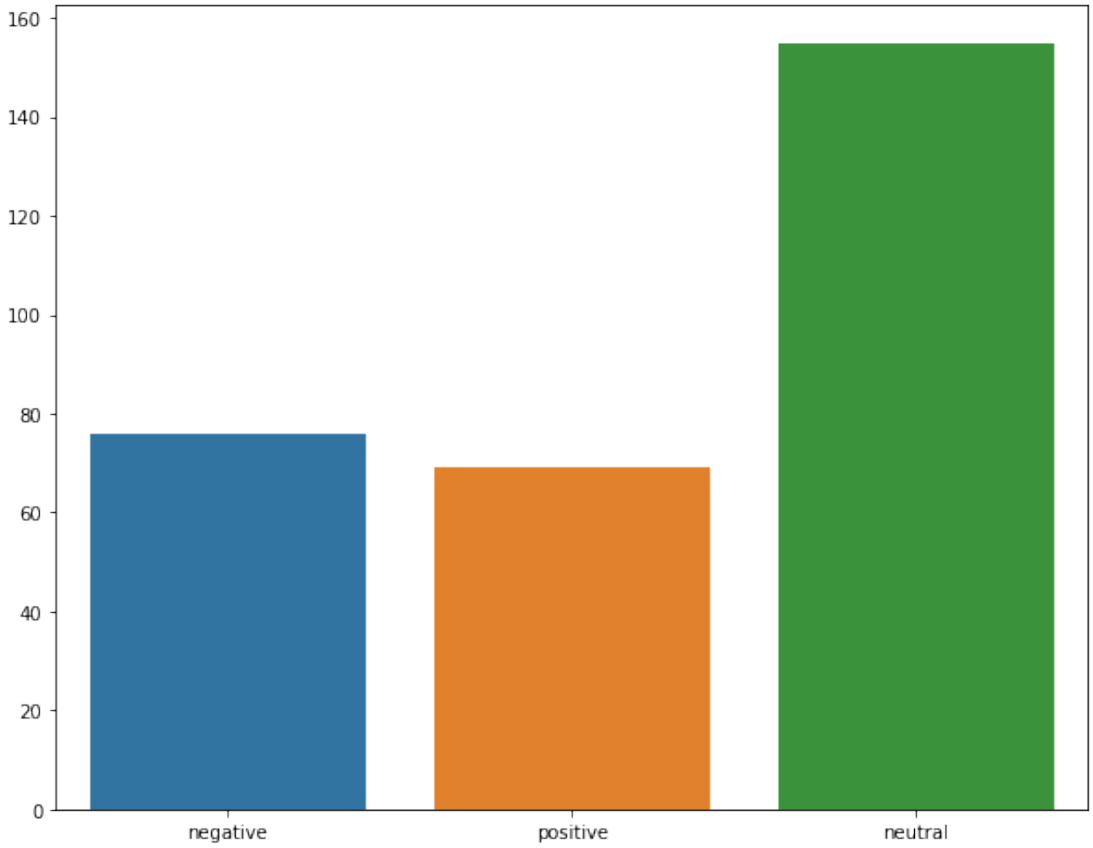
```
#format text for clear output
def formatText(text):
    formatted_text = ' '.join(word for word in text.split() if (word[0]!='#' and
                                        word[0]!="@" and
                                        not word.startswith("http") and
                                        not word.startswith("RT")))
    return formatted_text
```

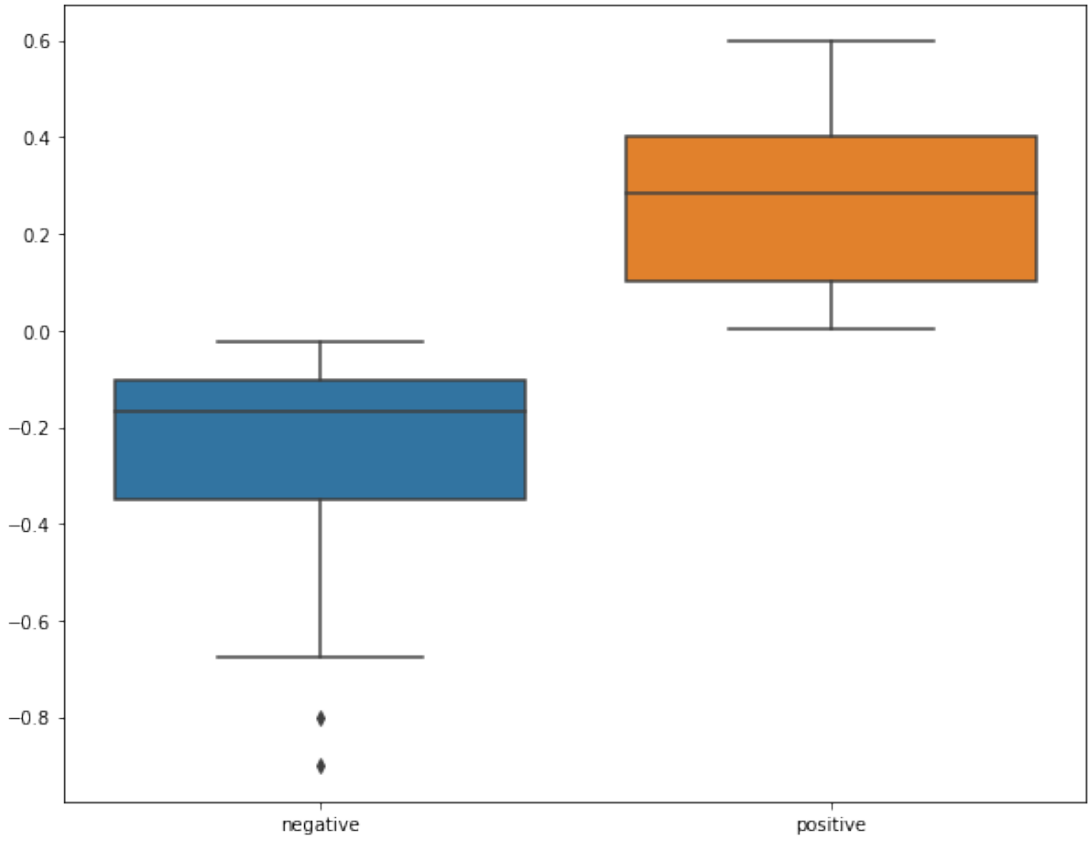
The following code chunk presents the procedure we used to categorize the sentiment of a tweet. If the polarity of the tweet is greater than 0 we assign it to the ”positive” class and if it is smaller than 0 we assign it to the ”negative” class. For tweets that have a polarity of exactly 0, we assume that they do not convey any sort of sentiment and categorize them as ”neutral”. Our application also prints the text of the first five tweets with their respective sentiment values from the textblob function for demonstration purposes.

```
while i < len(sen):
    #compute sentiment analysis on clean text and store it in opinion
    opinion = TextBlob(formatText(sen["text"][i]))

    #print 5 tweets with sentiment analysis
    if(s <= 5):
        print("For the text: ", formatText(sen["text"][i]))
        print("The text polarity is: %.2f" %opinion.sentiment[0])
        print("The text subjectivity is: %.2f" %opinion.sentiment[1])
        print("**************************************************")

    #store tweets in positive/negative/neutral polarity list
    if(opinion.sentiment[0] > 0):
        #add 1 to positive count because sentiment was positive
        countPosNegNeu[1] += 1
    elif(opinion.sentiment[0] < 0):
        #add 1 to negative count because sentiment was negative
        countPosNegNeu[0] += 1
    else:
        #add 1 to neutral count because sentiment was neither negative nor positive
        countPosNegNeu[2] += 1
    i += 1 #count +1 for while loop
    s += 1 #count +1 for text printing
```

Finally a plot is generated that visualizes the ratio between positive, negative and neutral tweets regarding the topic of interest. This should provide an anchor for understanding the polarity of sentiment present in the current discourse on Twitter.
Additionally a boxplot is created, that shows how positive the values of the positive tweets are on average and, of course, also how negative the negative ones are. These plots are also updated with new data every minute.

```
#print positive/negative/neutral barplot
print("Positive/Negative Barplot")
plt.figure(figsize = (10, 8))
sns.barplot(x=countNames, y=countPosNegNeu)
plt.show()

#print positive/negative/neutral boxplot
print("Positive/Negative/Neutral Boxplot")
boxplotdata = pandas.DataFrame(list(zip(negativeList, positiveList)), columns =['negative', 'positive'])
plt.figure(figsize = (10, 8))
sns.boxplot(data = boxplotdata)
plt.show()
```


160
140
120
100
80
60
40
20
0
negative
positive
neutral


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
negative
positive

2.4.3 Key System Features

One obvious key feature of Apache Spark we are using in our implementation is real time streaming. For our implementation and in order to process the tweets we need to manipulate real time data. Interestingly, for Spark this stream of data does not appear as one because of the batch intervall we set. This is meant by discretized stream, the basic abstraction in Spark Streaming. DStream is basically a series of RDDs to process our real time data.

So we can take the advantage of Spark that this system allows us to use the data core operations which are performed on the RDDs. After that, the processed data are pushed to the visualization part of our implementation.

This goes hand in hand with Spark's lazy evaluation and speed. As there is the read-only collection of objects partitioned across a set of a rebuildable machine we can work on the same data set multiple times: we can both count the hashtags an do the sentiment analysis on the same RDD. As mentioned earlier, the size of the data sets vary a lot because of e. g. the time during a week. If the data does not fit in memory Spark has the opportunity to perform operation external as it works also on-disk. As the goal is that performance is high, Spark will try to work as much as possible in memory.

A further key feature is the support of multiple languages. We could implement the application in python and in the official documentation there are also translations of the functions we used.

2.5 Problems Encountered

1. **RDD manipulation and debugging**: One major problem we discovered was the proper debugging of the RDD's of Spark. The problem here was, that you have to do the manipulation before you can see the output file because in our example it is a Streaming file after all. So every time we wanted to change something, we had to run the whole application to see whether it worked or not. Also there is no "general Cookbook Recipe" for RDD manipulation. One little change in code can mess up your whole implementation. Therefore it is really complicated and time consuming to work with them.
2. **Working with Tweepy**: In general Tweepy is the go to library if one wants to get and manipulate twitterdata in real time. The downside is, that the documentation of this package is not that well done. Also there is the problem that you have to stream the code in another file or shell simultaneously as you get it in your python file. Because we wanted to write our python code in jupyter notebook, we had to write a shell-file to execute the streaming pipeline with tweepy separately.

2.6 Alternative Implementation

Of course there are always many different ways to build an application. Among several opportunities, another machine which is also able to process real time data is Apache Flink.
To understand the difference between the two machines, it helps to compare the basic idea. Apache Spark offers streaming support based on batching. This means that Spark takes a portion of the incoming stream, pretends to have a small batch problem, solves it, and proceeds to the next batch chunk. Naturally, this is accompanied by a certain latency.
This is what Apache Flink tries to solve. Instead of making streaming a sequence of batching, the Flink batch task is a finite stream, thus turning the approach to processing streaming data around. It is clear that Flink can have an advantage in the need for fast processing of streaming data for decision making, as it provides higher performance.
This would be the case, for example, if in our application scenario a user demands a very fast decision making based on the mood analysis for his business field, because he wants to show his advertising directly on tweets only with a positive mood. In this particular case, the faster the decision is made, the faster an advertisement can be placed so that more people can see it. However, our own motivation for this project was to get to know a machine that is most commonly used in companies today, and our impression is that this is still Apache Spark.

References

[1] http://docs.tweepy.org/en/latest/code_snippet.html.

[2] https://mapr.com/getting-started-apache-spark/assets/getting-started-apache spark.pdf.

[3] https://medium.com/@naomi.fridman/install-pyspark-to-run-on-jupyter-notebook-on-windows 4ec2009de21f.

[4] https://spark.apache.org/docs/latest/api/python/index.html.

[5] https://spark.apache.org/docs/latest/index.html.

[6] https://towardsdatascience.com/hands-on-big-data-streaming-apache-spark-at-scale fd89c15fa6b0.

[7] https://www.earthdatascience.org/courses/use-data-open-source-python/intro-to-apis/twitter-data-in python/.

[8] https://www.geeksforgeeks.org/twitter-sentiment-analysis-using python/.

[9] A. Pavlo and M. Aslett. What's Really New with NewSQL?, in acm sigmod record. https://dl.acm.org/doi/10.1145/3003665.3003674, 2016. Accessed: 2020-05-05.

[10] M. Stonebraker. New Opportunities For New SQL, in communications of the acm. https://cacm.acm.org/magazines/2012/11/156577-new-opportunities-for-new-sql/fulltext, 2012. Accessed: 2020-05-05.

[11] W. Vogels. Blog post on Eventually consistency, in communications of the acm. https://dl.acm.org/doi/10.1145/3003665.3003674, 2009. Accessed: 2020-05-05.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.