MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

LICENSE THESIS TITLE

LICENSE THESIS

Graduate: Firstname LASTNAME
Supervisor: scientific title Firstname LASTNAME

2015

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Firstname LASTNAME**

LICENSE THESIS TITLE

1. **Project proposal:** *Short description of the license thesis and initial data*
2. **Project contents:** *(enumerate the main component parts) Presentation page, advisor's evaluation, title of chapter 1, title of chapter 2, ..., title of chapter n, bibliography, appendices.*
3. **Place of documentation:** *Example*: Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants:**
5. **Date of issue of the proposal:** November 1, 2014
6. **Date of delivery:** June 18, 2015 *(the date when the document is submitted)*

Graduate: _____

Supervisor: _____

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) _____, legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării _____
_____
_____
elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data _____

Nume, Prenume _____

Semnătura

De citit înainte (această pagină se va elimina din versiunea finală):

1. Cele trei pagini anterioare (foaie de capăt, foaie sumar, declaraţie) se vor lista pe foi separate (nu faţă-verso), fiind incluse în lucrarea listată. Foaia de sumar (a doua) necesită semnătura absolventului, respectiv a coordonatorului. Pe declaraţie se trece data când se predă lucrarea la secretarii de comisie.

2. Pe foaia de capăt, se va trece corect titulatura cadrului didactic îndrumător, în engleză (consultaţi pagina de unde aţi descărcat acest document pentru lista cadrelor didactice cu titulaturile lor).

3. Documentul curent **nu** a fost creat în MS Office. E posibil sa fie mici diferenţe de formatare.

4. Cuprinsul începe pe pagina nouă, impară (dacă se face listare faţă-verso), prima pagină din capitolul *Introducere* tot aşa, fiind numerotată cu 1.

5. E recomandat să vizualizaţi acest document şi în timpul editării lucrării.

6. Fiecare capitol începe pe pagină nouă.

7. Folosiţi stilurile predefinite (Headings, Figure, Table, Normal, etc.)

8. Marginile la pagini nu se modifică.

9. Respectaţi restul instrucţiunilor din fiecare capitol.

Contents

Chapter 1

Introduction - Project Context

Not long time ago, big data and the need for content analysis were idealistic, intangible concepts. However, recent developments in computer performance and the growing popularity of social media has enhanced the need for novel academic approaches in this field.

The biggest challenge with processing these huge amounts of data lies in its sheer volume. This is highly problematic in many different ways:

- storing and managing data is costly and sometimes needs special structuring attention
- irregularities across data streams often indicate the need for extra transformations
- filtering out noise and the extraction of meaningful data from thousands of entries requires specialised algorithms, architectures and platforms which are either inaccessible or difficult to use by regular users

1.1 Project context

Social media platforms these days have a strong presence in ”the Cloud”, which allows more and more developers to solve the problem of storage space and scalability in a relatively easy way. The high adoption of platforms like Facebook, Twitter, Instagram and many others have resulted in the generation of hundreds of thousands of online documents, text and media alike. Such a deluge of information is highly applicable for business purposes, but in is often stuck as a ”diamond in the rough”. An array of new and exciting job titles has also appeared, such as ”Social Media Marketer”, ”Social Media Manager”, ”Social Media Expert” and so on. Not only do they represent career alternatives for a new generation, but they are also still outliers for decision support software.

1.1.1 Social Media and why people care about what it says

While the concept of Social Media is not a new one to be precise, it is clearly just starting to engulf us. The spread of Social Media alongside the spread of worldwide Internet access, the emergence of specialised applications and the recent shift to mobile online presence are objective realities. Subjectively as well, people have become more engaged to consuming, creating and sharing content online, which is evident in newspaper sales drops, app sales increase, events "going viral" within groups and even the online presence of a political critical mass. Companies offer barter possibilities for a variety of bloggers and vloggers to push products to their followers (which is not only more effective[1], but also cheaper).

No wonder social media has a stake in public decisions! But for the first time in decades, people's choices are also quite open and public. Which means that some parties can get clues about society's preferences by using public data available on platforms such as Facebook, Youtube and Twitter. Consider two basic use cases, which I will emphasise throughout the course of this thesis:

- What does social media say about X?
- What does social media say about X vs. Y?

where X and Y can be public people, events, brands, companies etc.

1.1.2 Data Analysis tools today

There are a number of current platforms for data analysis, in the large sense including chart generators, format converters and importers and data visualisation plug-ins of sorts. Crossing into the realm of powerful analysis applications, most of them are complex and confusing to non-engineers, examples including Tableau[2], RapidMiner[3] and WolframAlpha[4]. Figure 1.1 shows Tableau's interface, which may seem quite intimidating.

1.1.3 Crossroads: Social Media in Business Intelligence

The Computer Science field is very much branched nowadays in clearly-delimited subjects. But it is such a case where the large-scale processing capacities of data analytics should merge with approaches for computer-assisted business decisions. Consider that data analytics software often requires extensive training and a specialised knowledge, while most users desire only simple, straightforward functionalities for basic data analysis. One

[1] http://www.techrepublic.com/article/election-tech-why-social-media-is-more-powerful-than-advertising/

[2] https://public.tableau.com/s/

[3] http://rapidminer.com

[4] http://www.wolframalpha.com

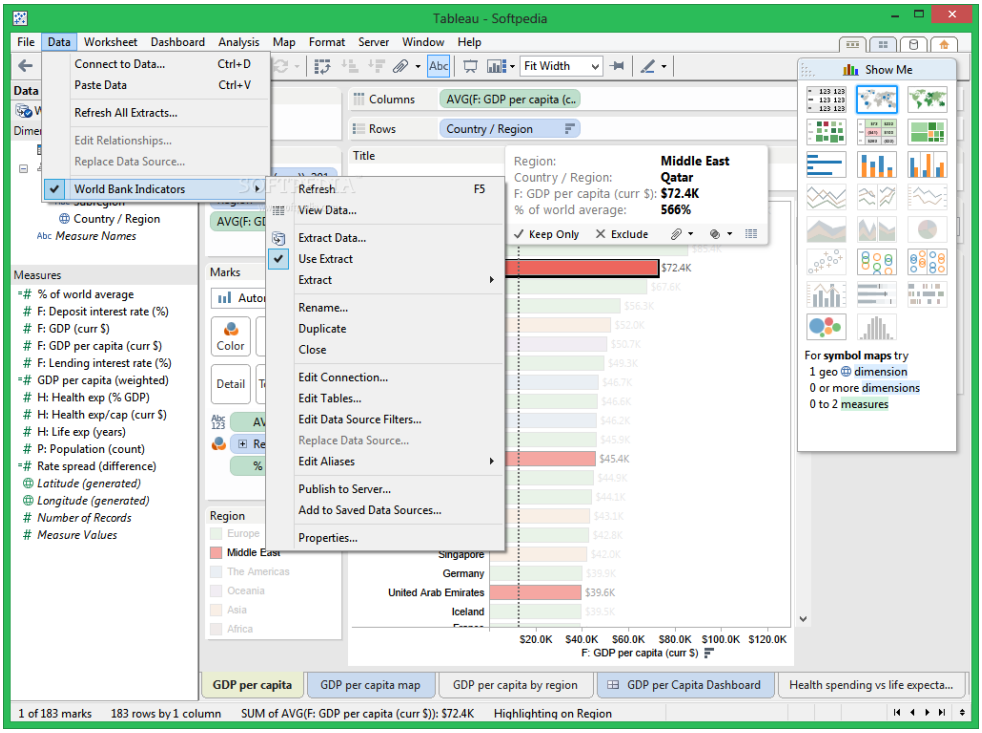

Figure 1.1: Tableau's interface

of Object Oriented Programming's fundamental principles is that of abstraction, in which end users need not (and should not) be aware of what is under the hood. Yet in data analytics, this is still the case on a

The following Master's Thesis is the result of such an endeavour, to combine solid concepts from big data analysis, web application architecture and design, as well as enterprise software and economics for a better understanding of social media trends and opinions.

Thesis structure

TODO

Chapter 2

Project Objectives and Specifications

The project's purpose is to help non-expert users interested in social media analysis to get relevant and reliable information regarding specific concepts.

2.1 Web architecture

A web architecture is preferable as a modern-day alternative to installable applications. The fast evolution of mobile devices has pushed web developers to segregate User Interfaces from Backend implementations, permitting devices to seamlessly connect to the same under the hood implementation without much development cost and overhead. Communication via REST services is crucial in this concern. Beyond user preference towards such applications, a web architecture presents even more advantages:

- virtually infinite space extensible with storage backends such as AWS3 or Google Cloud
- scalability and outsourcing of time-costly services such as getting stream histories
- database distribution over nodes
- easily available documentation and integration throughout the development process

Using a web framework eases development even more so and presents security advantages such as stable features, packages and quick patches in the odd case of a security breach. Plugging in services is also considered in the project's context, with handling of specific use cases done in separate managers, loosely coupled with the Model-View-Controller architecture.

2.1.1 Data storage

Since scalability and distribution are a must for a large-scale stream analysis project, data storage should be handled using a method which supports such breakdown while still remaining query-efficient. NoSQL databases are a popular choice for such implementation. The correspondence between the database storage backend and the model part of the application should also be loosely coupled, allowing for the possibility of switching between database backends if the need arises.

Big players in the field of non-relational databases are obvious possible choices: HBase, Cassandra, MongoDB, CouchDB, etc.

2.2 ATHENA breakdown

This thesis is based on the ATHENA original project, which comprises different pipeline steps in acquiring and analysing Twitter feeds:

- Harvesting: acquisition of collections of related Twitter statuses, by hashtag and dates (start and end).
- Enhancement: basic analysis of a single harvest with classical unsupervised algorithms
- Normalisation and Analysis: comparative analysis of harvests

2.2.1 Twitter as data source

For the development of this project, I chose to only implement one social media backend as data source. Twitter was the choice, since the character limit they impose is a great asset in regards of data processing: First of all, the classical data analysis pipeline deals with special cases such as documents of different lengths providing unusual skewing to the data set. Another advantage of Twitter is the prevalence of the hashtag model, which has failed to catch on with Facebook to the same degree. This means that statuses basically come out of the box already tagged for content.

Twitter is arguably the second most popular social media platform at the time, gaining on a large number of users even before the mobile device explosion, by integrating with telephone service providers to facilitate posting through SMS. Its 140 character limit makes for a good spin, which has pushed Twitter into a more textual realm than its more successful counterpart, Facebook. The platform is largely popular in the United States and especially in the entertainment domain[1], with the most popular Twitter accounts belonging to celebrities.

[1] https://en.wikipedia.org/wiki/Twitter

It notably also has a history of being the ”go to” source for data analysis, after the United States' current president Barrack Obama purportedly used it in his 2008 campaign as one of the principal media outlets, choosing to have a strong Twitter presence, akin to advertisement presence of past candidates.

In short, although this approach is social platform-independent, Twitter was chosen even from the specifications step of this project for a clear start towards the data mining part.

Fetching data through asynchronous jobs

Since most social media platforms communicate with external applications via REST APIs with rate limits, the project's structure should include asynchronous modules. In this way, the user will be able to send an API-intensive job for queueing and asynchronous processing, without hindering their overall experience of using the application. This becomes apparent in the Harvesting part.

2.2.2 Harvesting

If we interpret ATHENA as a classical pipeline, then it becomes clear that the Harvesting part is the acquisition part. In order to perform analysis on data, it must first be fetched according to predefined rules and user options. Being time-intensive, these jobs must be designed asynchronously, following a FIFO model.

Figure 2.1 represents the conceptual model of data acquisition, starting from a collection of grouped documents (in our case they will be collections of statuses with a common hashtag). The Harvest manager handles sending asynchronous jobs via a FIFO queue towards a Consumer-type Harvesting job, represented as black box and further detailed in Figure 2.2. It is important to note that this pipeline is not in any way adjusted to the domain or the particular API platform; the only constraint we impose is that of periodically checking for compliance to rate limits. Social APIs usually limit requests either by number of requests or requested data size per unit of time. It is therefore a good idea to consider if any limitations alter our pipelines before starting on implementations.

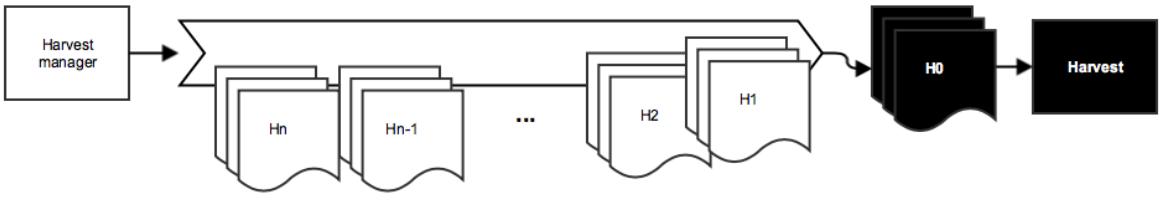

Figure 2.1: Harvesting pipeline

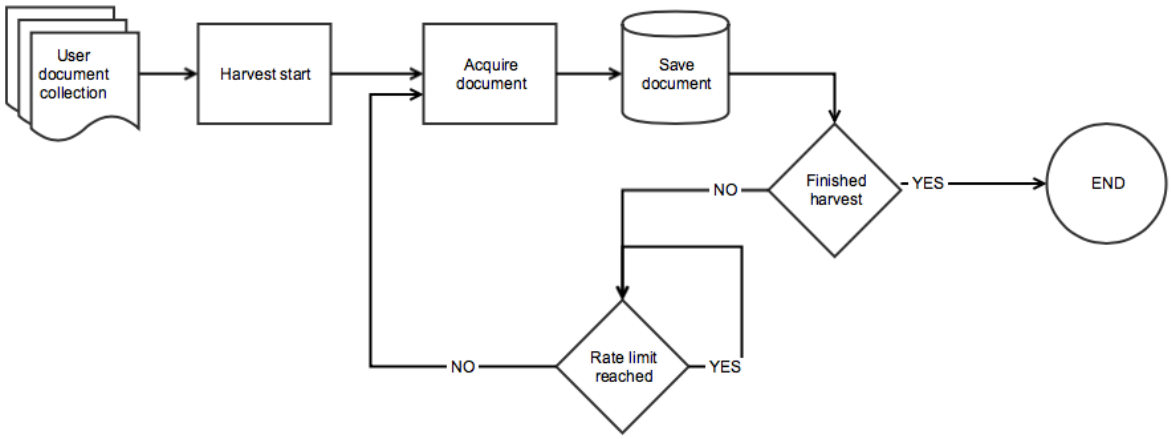

Figure 2.2: Harvesting job sub-pipeline

2.2.3 Enhancement

As previously stated, it is not of great importance to have the data, but the goal is to obtain meaningful information from it. In this thesis I will refer to ”harvests” as per the following definition:

Definition A Harvest is a collection of documents related by contant, containing a token T and spanning from a start date and to an end date, which have gone through the harvesting pipeline and are stored for further anaylsis.

After completing the Harvesting pipeline, resulting harvests are processed using various data extraction algorithms and presented to the user in the Enhancement step. These algorithms each have their own sub-pipelines of transformation, which will be detailed in the Analysis step.

User contribution modeling

TODO

2.2.4 Normalisation and analysis

TODO

2.3 Non-functional requirements

I am much concerned with the user's perspective in developing this application. As explained previously, one of the main objectives of this application is undoubtedly to be as simple to use as possible, to benefit non-expert users.

Firstly, I believe that a familiar and streamlined user experience should be implemented. The UI should have componenents found in most web application nowadays, in order to seem aproachable. The customisation options should also be loaded without input from the user, so that things like algorithm choices and parameters should stay hidden. Non-expert users do not know what *Tfidf Vectorizer* or *KMeans clustering* means, all they care about is to have some paplable results to their queries. In short, it is in the benefit of the non-expert users, which I target through this application, to have as little inputs, panels and buttons to figure out.

Besided the look and feel, users will also be interested in having a fast and reliable application. Exceptions and possible bugs should be properly filtered out and time-consuming jobs moved as per Figure 2.2 to asynchronous jobs.

Chapter 3

Bibliographic research

Bibliographic research has as an objective the establishment of the references for the project, within the project domain/thematic. While writing this chapter (in general the whole document), the author will consider the knowledge accumulated from several dedicated disciplines in the second semester, 4^{th} year (Project Elaboration Methodology, etc.), and other disciplines that are relevant to the project theme.

Represents about 15% of the paper.

Each reference must be cited within the document text, see example below (depending on the project theme, the presentation of a method/application can vary).

This section includes citations for conferences or workshop [**?**], journals [**?**], and books [**?**].

In paper [**?**] the authors present a detection system for moving obstacles based on stereovision and ego motion estimation. The method is ... *discus the algorithms, data structures, functionality, specific aspects related to the project theme, etc....* Discussion: *pros and cons.*

In chapter 4 of [**?**], the *similar-to-my-project-theme algorithm* is presented, with the following features ...

3.1 Title

3.2 Other title

Chapter 4

Analysis and Theoretical Foundation

TODO: Producer consumer architecture

Together with the next chapter takes about 60% of the whole paper

The purpose of this chapter is to explain the operating principles of the implemented application. Here you write about your solution from a theory standpoint - i.e. you explain it and you demonstrate its theoretical properties/value, e.g.:

- used or proposed algorithms
- used protocols
- abstract models
- logic explanations/arguments concerning the chosen solution
- logic and functional structure of the application, etc.

YOU DO NOT write about implementation.

YOU DO NOT copy/paste info on technologies from various sources and others alike, which do not pertain to your project.

4.1 Title

4.2 Other title

Chapter 5

Detailed Design and Implementation

The following chapter presents the architecture, components and implementation particularities of the ATHENA application.

5.1 Python and Django

The choice of Python as main programming language for this application, alongside the choice of Django as a framework, were based on the industry's long-time endorsement of these technologies. According to the TIOBE programming language index, Python ranks as the 6th most popular programming language, the index being calculated according to the number of engineers world-wide, courses and third-party vendors concerned with this programming language [1]. Its higher ranked competitors are Java and the C family (C, C++ and C#), which are not as well diversified in the field of research and string processing as Python.

Python for scientists

Besides its prevalence in the web application realm, Python is also one of the preferred languages for research, as argued in [2], with a plethora of scientific-oriented third-part libraries including NumPy, SciPy and Matplotlib. String processing is also generally considered faster and more cohesive than in other programming languages.

Python libraries

One of the main libraries used here is Django, arguably the best known Python web framework. Django was chosen in the detriment of others such as Flask and CherryPy, due to its active and dynamic open source community, its good documentation resources, high number of compatible third party libraries and generally available support with installation, development and running.

For the purpose of this application, a number of libraries were installed via `pip`, the package manager preferred by Python applications. The entire application was build using `virtualenv`, a virtual environment creator that helps Python developers manage different Python versions in different environments, to avoid unnecessary creation of virtual machines. Running the command `pip freeze` inside the activated virtual environment we get the list of installed third party libraries:

```
amqp==1.4.9
anyjson==0.3.3
beautifulsoup4==4.4.1
billiard==3.3.0.23
cassandra-driver==3.4.1
celery==3.1.18
cssselect==0.9.1
cycler==0.10.0
DistributedLock==1.2
Django==1.8.13
django-annoying==0.9.0
django-m2m-history==0.3.5
django-oauth-tokens==0.6.3
django-picklefield==0.3.2
django-taggit==0.19.1
futures==3.0.5
kombu==3.0.35
lxml==3.6.0
nltk==3.2.1
numpy==1.11.0
oauthlib==1.1.1
pandas==0.18.1
pyparsing==2.1.4
pyquery==1.2.13
python-dateutil==2.5.3
python-memcached==1.58
pytz==2016.4
redis==2.10.3
requests==2.10.0
requests-oauthlib==0.6.1
scikit-learn==0.17.1
scipy==0.17.1
simplejson==3.8.2
six==1.10.0
sklearn==0.0
tweepy==3.5.0
```

The advantage of using the `pip` package manager is that upon installing either library, its dependencies are installed as well. Throughout this thesis I will refer to the list of installed libraries when explaining the choice and particular implementation where each library is used. For now, please note the Django 1.8 installation, which is the current long-term release, chosen for its stability and support.

5.2 Storage with Cassandra

The Cassandra NoSQL database was chosen for this project's non-relational database requirement, due to its large flexibility and scalability to more clusters when the need arises.

A Cassandra server was installed locally and needs to be up for all queries run

from the aplication. The Python library `cassandra-driver` allows for easy connection to Cassandra clusters and execution of queries, similarly to SQL ones. Empyrical debugging is easy via the `cqlsh` command line utility, which acts like a Cassandra interactive console. Figure 5.1 presents a demo of these functionalities, including starting up the console, connecting to a cluster and submitting a query.

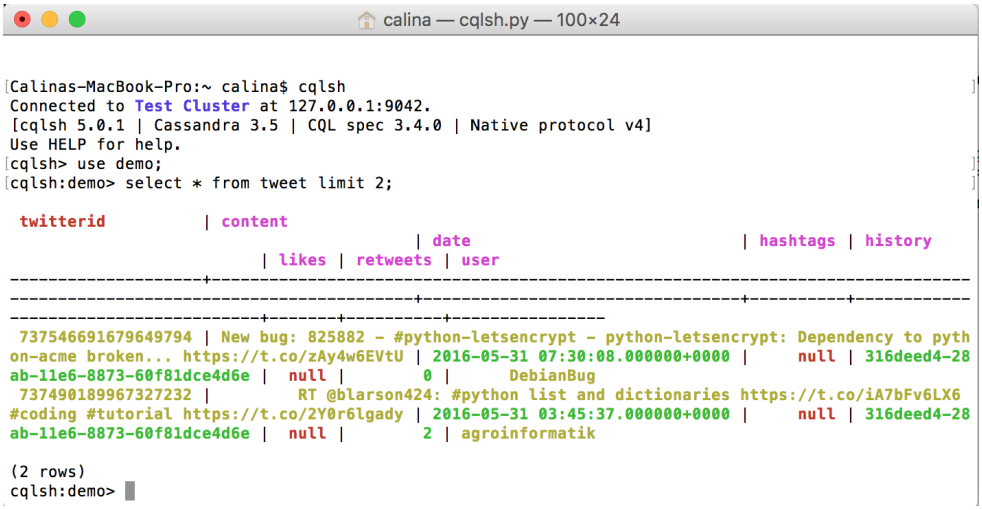

Figure 5.1: Demo of the cqlsh utility

The same functionality can be mirrored in Python using the following set of instructions:

```
from cassandra.cluster import Cluster

cluster = Cluster()
session = cluster.connect('demo')

harvests = session.execute(
    """
    select * from tweets limit 2
    """
)
```

with the result being a generator which can be further consumed by the application. This means that it is easy to perform queries programatically, without much overhead, by employing the usage of this driver.

Cassandra queries are used throughout the application, both in synchronous and asynchronous application steps. Its speedy retrieval and updating of records is not a bottleneck, as API rate limits are.

5.2.1 Database structure

Two tables are used in our application. The `harvest` table containing columns:

- uuid (primary key, generated for each user-submitted harvest form)

- start date (from which we harvest Tweets)
- end date (up to which we harvest Tweets)
- hashtag (used to query Twitter's API for statuses containing this particular hashtag)
- done (boolean fag indicated whether the harvest has finished downloading Tweets)

The `tweet` table is used for storage of tweets belonging to histories and contains:

- twitterId (the id of that status on Twitter)
- user (username of the author on Twitter)
- content (textual content including hashtags)
- date (of postage)
- retweets (number of retweets, indicating popularity)
- history (uuid of harvest that Tweet belongs to inside ATHENA)

5.3 Harvesting tools

The harvesting module as presented conceptually in 2.2 was implemented in ATHENA using asynchronous job queueing. The user is presented with a form for submitting the harvesting job, consisting of a content field, start and end dates. An asynchronous job is launched via Python's Celery task library, which uses the tweepy library to connect to the Twitter Search API and Cassandra driver to save tweets and harvests.

5.3.1 Celery for asynchronous jobs

The Python library Celery[1] is designed for the fairly frequent development case where asynchronous jobs need to be managed. The Producer-Consumer architecture is implemented with Celery, as used in its real-time mode, while another option would be using Celery to run scheduled jobs. The job granularity in this case is indeed Harvest-level, with each job inside the Producer-Consumer queue is defined as the job of downloading one single, separate Harvest.

As previously stated, a huge advantage of Python is that third party libraries are highly compatible and easily configurable. Such is the case with Celery as well, setting up an asynchronous jobs taking very little effort:

1. install the `celery` library using `pip`

[1] http://www.celeryproject.org

2. add configuration information to a celery.py file inside the application directory, including the task body and its decorator `@app.task(bind=True)`

3. import the function and use it. In the ATHENA Harvesting module context, the function was added as part of the Form validation customisation in Django's FormView class

4. install and configure a service broker such as Redis

5.3.2 Using Redis as a Celery broker

Celery offers a variety of possible brokers for message transport through the job queue. Stable brokers are RabbitMQ and Redis, while others are in Experimental stages or offered by third parties[2]. Redis is an open source data structure store which can perform as a database or cache system, but in our case we are interested in its functionality as a message broker.

Installing Redis is straightforward using a downloaded package and even some available package managers such as Mac's `brew`. After the installation is complete, the Redis server can be fired up using the command `redis-server`. A splash screen with Redis' logo as ASCII art should appear, but connection to the running Redis server can also be tested by pinging:

```
$ redis-cli ping
PONG
```

After the Redis server is properly installed and running, there are some extra steps for hooking it up: adding the Redis configuration settings in our project's `settings.py` file and installing the `redis` library using pip. After all these steps are completed, running:

```
celery -A athena_app worker -l info
```

should confirm Celery's connection to Redis. Any tasks that were previously set up will now go through this job queue.

5.3.3 Fetching data from Twitter using the Search API and Tweepy

Once the general configuration of the asynchronous job is done, we can use the Twitter Search API to Harvest tweets per the specifications.

Twitter belongs to a series of web applications which fully understands the developers' need to hook into some of their features. Creating a Twitter application is easy from their developer support pages, with the creator receiving a set of OAuth access keys:

- a Consumer Key (API Key)

[2] http://docs.celeryproject.org/en/latest/getting-started/brokers/

- a Consumer Secret (API Secret)
- an Access Token
- an Access Token secret

For harvesting tweets, my approach uses a wrapper to Twitter's Search API called Tweepy. It is a library that handles connection and customised requests to the API, in a Pythonic fashion. Tweepy needs to be installed using pip, and then configured with the proper access keys (here in the code sample replaced with placeholders for security purposes):

```
from tweepy import OAuthHandler

consumer_key="XXXX"
consumer_secret="XXXXXXXX"

access_token="XXXX"
access_token_secret= "XXXXXXXX"

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

The way I am using this is by first setting up an `api` object which I will further query for statuses. I will initialise it using the `auth` object and a few flags:

- `wait_on_rate_limit=True`. Flagging this indicates to the api object that whenever it reaches the API rate limits, it should not stop, but rather sleep for the required amount of time until new data can be acquired. This approach is preferable, since our harvesting takes place asynchronously and we do not mind the process sleeping for a while
- `wait_on_rate_limit_notify=True`. Setting this flag tells the api object to print out a warning string to the console, indicating when it has reached the rate limit and at the beginning and end of the sleep cycle. E.g.:

  ```
  [2016-06-02 10:17:46,804: WARNING/Worker-2]
  Rate limit reached. Sleeping for:
  627
  ```

After setting up the API, a Tweepy Cursor class is used for querying it. This is initialised with the api object and parameters indicating the query we send to the server. This uses parameters defined by the user upon submitting the harvest form, refined by our application's specifications. The hashtag field is prefixed with a "#" sign, start and end dates are formatted to "yyyy-mm-dd" and the language set to English, since extension to other languages is beyond the scope of this thesis.

The cursor will return a Pyhton generator, which means that the contents of the tweet list is rather consumed one by one, rather than having the whole list present at either point in time. Here, using Cassandra, we save the contents to the database.

```
tweets = tweepy.Cursor(api.search, q='#' + hashtag, since=start_date,
    until=end_date, lang='en').items()

for tweet in tweets:
    session.execute(
     """
     insert into tweet (twitterId, user, content, date, retweets,
        history) values (%s, %s, %s, %s, %s, %s)
     """,
     (str(tweet.id), tweet.author.screen_name.encode('utf8'), tweet.
        text, tweet.created_at, tweet.retweet_count, key)
     )
```

The stored and completed harvests are now ready for the Enhancement step.

5.4 Enhancement tools

5.4.1 Simple numerical data

5.4.2 Sci-kit learn

5.4.3 Related hashtag calculation

5.4.4 Related hashtag clustering using K-Means

5.4.5 Page and bot vs. users modeling

TODO

5.5 Normalisation and Analysis tools

TODO

Chapter 6

Testing and Validation

About 5% of the paper

6.1 Title

6.2 Other title

Chapter 7

User's manual

In the installation description section your should detail the hardware and software resources needed for installing and running the application, and a step by step description of how your application can be deployed/installed. An administrator should be able to perform the installation/deployment based on your instructions.

In the user manual section you describe how to use the application from the point of view of a user with no inside technical information; this should be done with screen shots and a stepwize explanation of the interaction. Based on user's manual, a person should be able to use your product.

7.1 Title

7.2 Other title

Chapter 8

Conclusions

About. 5% of the whole
Here your write:

- a summary of your contributions/achievements,
- a critical analysis of the achieved results,
- a description of the possibilities of improving/further development.

8.1 Title

8.2 Other title

Bibliography

[1] "TIOBE index May 2016," http://www.tiobe.com/tiobe_index?page=index, 2016, [Online; accessed 5-June-2008].

[2] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

Appendix A

Relevant code

```
 /** Maps are easy to use in Scala. */
object Maps {
  val colors = Map("red" -> 0xFF0000,
                   "turquoise" -> 0x00FFFF,
                   "black" -> 0x000000,
                   "orange" -> 0xFF8040,
                   "brown" -> 0x804000)
  def main(args: Array[String]) {
    for (name <- args) println(
      colors.get(name) match {
        case Some(code) =>
          name + " has code: " + code
        case None =>
          "Unknown color: " + name
      }
    )
  }
}
```

Appendix B

Other relevant information (demonstrations, etc.)

Appendix C

Published papers