Towards a Scalable Data-Intensive Text Processing Architecture with Python and Cassandra

Gregor-Patrick Heine, University of Applied Sciences Wr. Neustadt
Thomas Woltron, University of Applied Sciences Wr. Neustadt
Alexander Wöhrer, University of Vienna

Introduction

Approach Taken

Novel approach towards social media sentiment analyses via **streaming** live data.

Canonical approaches hinge on hashtag searches via the Representational State Transfer Application Programming Interface (**REST** API).

Our approach has both, **higher data recency** and a **clearer topic related** natural language **structure** by design.

Hypothesis

Twitter is a medium for public discussions.
The initial post (**headline**) introduces the **topic**.
Consecutive **replies** constitute the **body**.
Topics are defined by **nouns**.
Noun-phrase frequencies are the **consensus** on what is important in discussion.

More replies
>> higher noun-frequencies
>> more descriptive noun-phrases

Frequent noun-phrases
>> public consensus on topic (or headline)

Methods & Related Work

Text Processing

Natural Language Processing (NLP):

Text as input information

Decomposition via syntax and semantics

Source - Twitter:

140 to 280-character long tweets

Platform for political debates

Live streaming and mining Tweets:

Noun-phrases are indicative of opinions

Topic context is created on read

United States politics - binary reactions

Data Pipelining

Data Store:

Cassandra NoSQL database

Parallelization:

Moore's law is exhausted

(integrated circuits double every 3 years)

Amdahl's law more applicable

(faster when running code in parallel)

Data parallelization

Threads do same operations on separate items

Split data symmetrically

Task parallelization

Threads do different operations on different items

Atomize data and put into a queue

Multiprocessing

Processes do not share memory

Threads share both state and memory.
Code is scheduled by Operating System (OS)

Time division multiplexing
Illusion of running multiple threads in parallel when in fact, switching between threads quickly.

Python's Global Interpreter Lock (GIL):
Multithreading in Python is managed by the OS
GIL limits the number of running threads to one.
Prevents conflicting writes (race conditions)

Cassandra

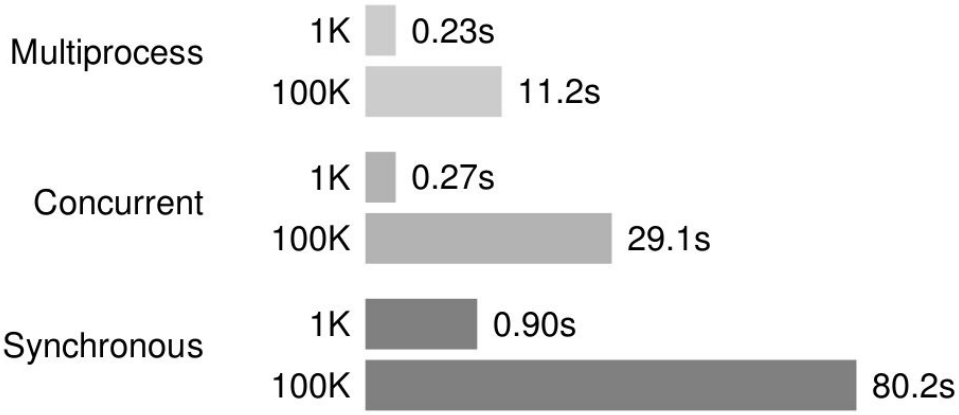
- Wide column store
- NoSQL database management system
- Designed to handle large amounts of data
- Across many servers
- Providing high availability
- Without single point of failure
- Staged Event-Driven Architecture (SEDA)

Implementation & Results

Batch Processing

1. Multiprocess Concurrent Inserts
2. Single Thread Concurrent Inserts
3. Single Thread Synchronous Inserts

Figure 2. Comparing elapsed seconds inserting records into Cassandra



Querying Data

- `GROUP BY` clauses are not supported
- `ORDER BY` supported implicitly via table declaration `WITH CLUSTERING` on PRIMARY KEY columns
- **Filtering** is supported but may cause malfunctions due to the nature of Sorted String Tables (SSTables) and Log-Structured Merge-Trees (LSM-Trees)
- **Iterate** returned values using Python lists
- **Tombstones** are deletion markers which also occur when inserting null values or collections.

Noun-Phrases

The number of **followers** indicates the **influential** weight of a post

Overall noun **frequencies** yield the public **consensus**.

Lower case converted **stemmed** word frequencies best illustrate topic related word convergence due to **minimizing differences** regarding, capitalization, misspellings and affixes.

Hypothesis Results

In order to get an overview of the public opinion related to the headline, it is recommended to **print** the six highest ranking **results**.

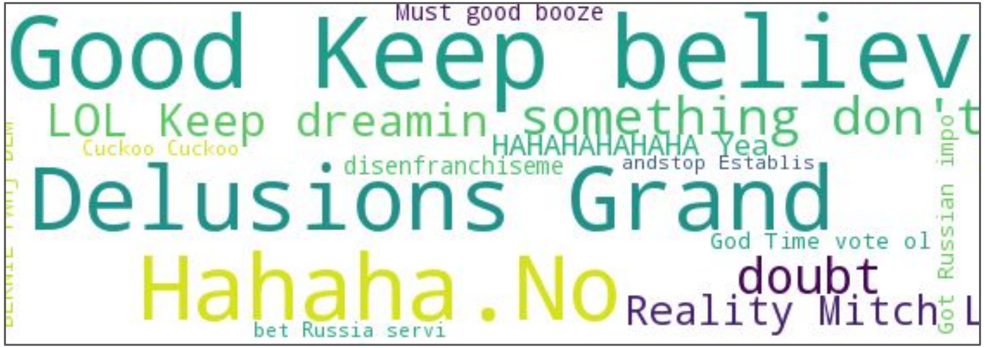
“Senate GOP:
We will grow our majority in midterms”

Preliminary results indicate that the general opinion is **ridicule**.

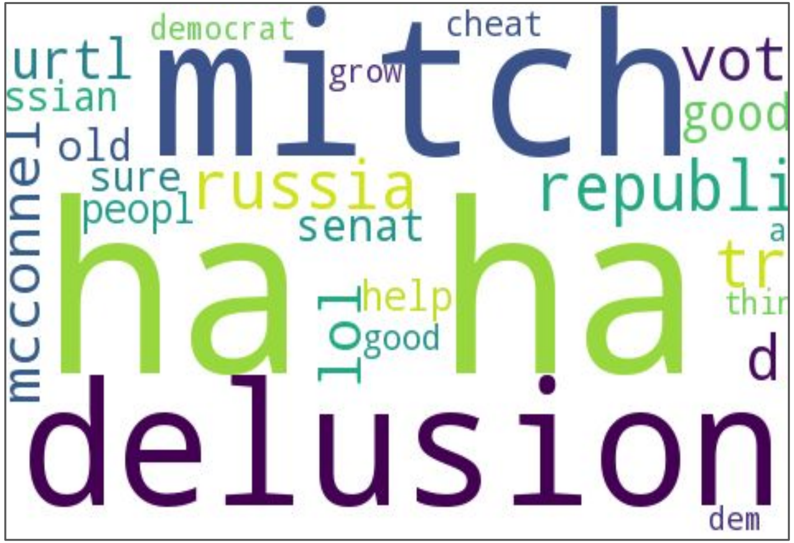
Little credence is given to Trump administration

Visualizations are performed via **word clouds**

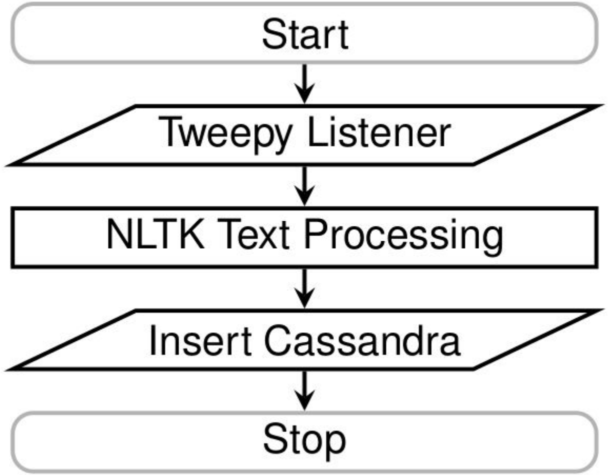
Noun-Phrases



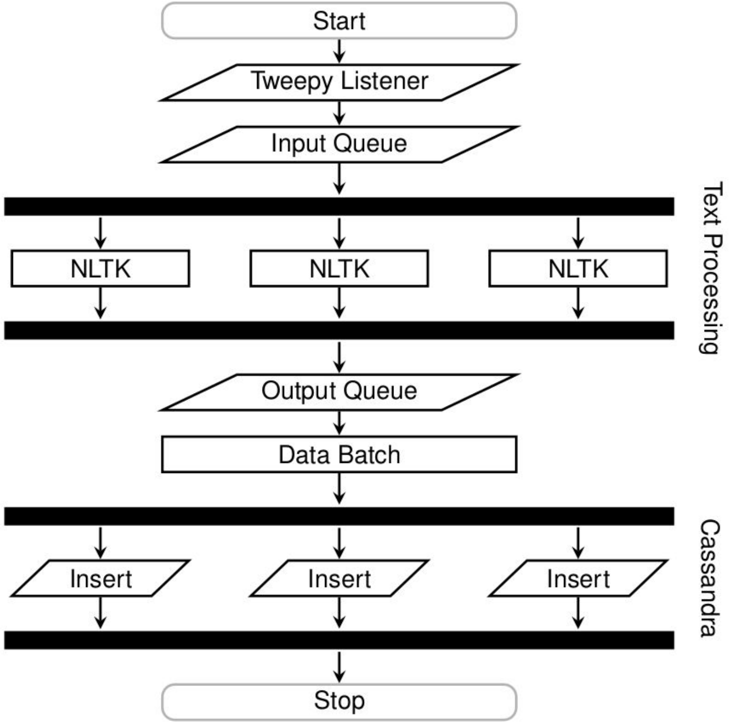
Word Frequencies



Sequential Processing



Parallelized Processing



Conclusion & Discussion

Cassandra & Python

Proposed application relies on:

1. Suitable **data sources**
2. Associated **use cases**
3. **Processing power** and **memory size**
4. Data **persisting** methods

Cassandra is advocated and its inner workings are illuminated.

A **scalable text processing** application is developed wholistically in a big data context.

Stream & Text Mining

Innovative and uncommon **solutions** are introduced.

Twitter **streaming** and **text mining** has **not been discussed** as such in literature before.

Noun-phrase and word frequency **convergences** are both reviewed and analyzed in a novel way.

Public debates are analyzed and persisted in (near) real-time without keyword restrictions.

Introduced methods could be employed for **public surveillance.**

Future Developments

Focus on **multithreading**

Employ Python's **multiprocessing package**

Java allows for better multithreading, which probably increases execution **speed**

Text processing in **Java** may be **cumbersome**

Migrating the project to the **GoLanguage** for native support regarding **parallelism** is worth pursuing.

Future Research

Monitoring both news and **stock prices**

Investigate which **news** cause what **reactions** and how this relates the price of an index.

Integration of (near) **real-time stock price** information in order to work with financial time-series data.

Dashboards and user experience enhancing charts are worth to be investigated.

Towards a Scalable Data-Intensive Text Processing Architecture with Python and Cassandra

Gregor-Patrick Heine, University of Applied Sciences Wr. Neustadt
Thomas Woltron, University of Applied Sciences Wr. Neustadt
Alexander Wöhrer, University of Vienna