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

Gregor-Patrick Heine
and Thomas Woltron

University of Applied Sciences Wiener Neustadt
Institute of Information Technology
Wiener Neustadt, Austria
Email: heine.gregor@gmail.com
Email: thomas.woltron@fhwn.ac.at

Alexander Wöhrer

University of Vienna
Faculty of Computer Science
Vienna, Austria
Email: alexander.woehrer@univie.ac.at

***Abstract*—Canonical sentiment analysis implementations hinge on synchronous Hyper Text Transfer Protocol (HTTP) calls. This paper introduces an asynchronous streaming approach. A method for public opinion surveillance is proposed via stream subscriptions. A prototype combining Twitter streams, Python text processing and Cassandra storage methods is introduced elaborating on three major points: 1) Comparison of performance regarding writing methods. 2) Multiprocessing procedures employing data parallelization and asynchronous concurrent database writes. 3) Public opinion surveillance via noun-phrase extraction.**

Keywords–Cassandra; Streaming; Python; Multiprocessing; Twitter; Sentiment Analysis

I. INTRODUCTION

Volume, Velocity and *Variety*, also known as the *3V*, generalize big data problems [1]. A fourth (fifth or sixth) V could be added referring to *Value, Variability* or *Virtual* [2]. Sticking to the more prominent 3V, *volume* naturally stands for a humongous amount of data. *Velocity* refers to the ingress, egress and process speeds. *Variety* is representative of heterogeneous data sets, which may be structured, semi-structured or unstructured. Essentially, the 3V break traditional data processing systems as they fail to scale to at least one of these attributes [3].

Big data has already found its place in business, society administration and scientific research. It is also believed to help enterprises improve their competitive advantage. Managers believe it to be a panacea and take a one-size fits all approach. It is also held in high regard with reference to aiding decision making processes [4].

From a high-level viewpoint, there are seven big data principles [4]: good architecture; variety regarding analytics; many sizes for many needs; perform analyses locally; distributed in memory computations; distributed in memory storage and process data set coordination.

However, without deeper understanding and practical knowledge they remain abstract buzzwords. Putting this into perspective by using an analogy of combustion engines, it becomes clear that it is impossible to repair, let alone build, a V8 (pun intended) by merely knowing the concept of thermodynamics. This paper outlines a practical big data streaming implementation in alignment with the seven principles.

Research Objective: The goal is to develop a scalable data-intensive application for text mining. Theory regarding sentiment analysis and opinion extraction are given in the working hypothesis. A naive architecture is depicted in Figure 1. In order to be able to monitor opinions, the following challenges must be tackled: high frequency real-time HTTP data-streaming; in-memory text mining and persisting results efficiently in a fault tolerant way.

Hypothesis: When following a publisher (or broadcaster) like 'The Hill' it is a matter of time until related public discussions start. The initial post is referred to as headline, which introduces the topic. All consecutive replies are regarded as the discussion's body.

Since topics are defined by nouns, their corresponding noun-phrases can be be listed. It can be argued that noun-frequencies represent a public consensus on what is deemed important in a topic related discussion. Hence a testable hypothesis can be formulated as such:

The more replies a headline receives the higher the number of noun-frequencies. The higher noun-frequencies the more descriptive are associated noun-phrases. When there are highly frequent noun-phrases, a clear public consensus regarding a topic specific headline exists.

We take a novel approach regarding 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.

The rest of the paper is organized as follows. Section II aims at demonstrating how high volumes of data, more specif-

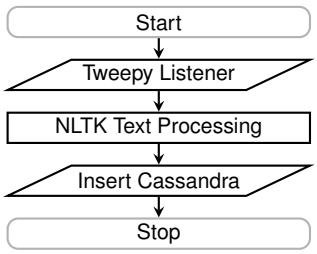

Figure 1. Sequential Python architecture under the Global Interpreter Lock

ically text, can flexibly and efficiently be processed in (near) real-time. Methods for text stream analyses and storage are also illuminated. Our main contribution is described in Section III where we proposes a scalable application architecture and provide a proof of concept implementation and evaluation. We close with our conclusions and future work suggestions in Section IV.

II. Methods & Related Work

Natural Language Processing (NLP): Text mining consists of "a vast field of theoretical approaches and methods with one thing in common: text as input information" [5, p. 1]. An introduction to the traditional field of NLP is quoted to be the "process of language analysis [that is] decomposable into a number of stages, mirroring the theoretical linguistic distinctions drawn between *syntax, semantics* and *pragmatics*" [6, p. 4]. Twitter, with its 140- to 280-character long tweets (or posts) promises to be a rich medium for scientific analysis [7]. Overall, Twitter can be regarded as a platform for political debate [8] and enables measuring the public opinion with regards to politics [9].

Nouns and noun-phrases are likely to converge towards a dominant *thesaurus*. Hinging on frequent nouns and noun-phrases it can be concluded that they are extracted from text. Hence, they are countable and highly frequent noun-phrases can be deemed important. This approach is utilized in the context of online product reviews and is argued to be domain dependent. Essential to this approach is Part-of-Speech (POS) tagging [7].

Mining Tweets: Noun-phrases are indicative of opinions as they use adjectives and adverbs together with nouns. When live-streaming tweets there is no a priori means for clustering them. A topic's context is created retrospectively when reading records from a data store.

News agencies concerned with United States politics are likely to evoke binary reactions. Opinion holders are either in favor or against Democrats or Republicans. Following that train of thought, it can be assumed that neutral political headlines need to be immediately digested and either be supported or opposed.

When tweets are received they cannot be put into context right away. They have to be persisted for later retrieval. While streaming, it is advisable to process text immediately, since it is already held in memory. This allows in memory pointer manipulation, regarding tokenization, Part-of-Speech tagging, chunking and parse-tree creation. All of which are needed for later analysis. When reading from the database, a post's meaning arises through its sequential context. Reading preprocessed noun-lists and iterating them in order to create word frequencies is faster than processing text in bulk after reading.

Parallelization: It appears that computer science has exhausted *Moore's law* as integrated circuits only seem to double every three years. Today, the real performance boost comes from running multiple threads simultaneously. This is also referred to as Thread-Level Parallelism (TLP) [10]. *Amdahl's law* [11] has become more prominent in recent years, it states that program execution time can be improved through running code in parallel [12]. It needs to be noted that this law cannot be exploited *ad infinitum* as it approaches a point of diminishing returns.

Parallelization is a powerful means of increasing processing speed. It needs to be noted that writing parallel code is much harder than writing serial code. However, generally speaking, programmers face a choice between two parallelization paradigms: *data parallelization* has multiple threads performing the same operation on separate items of data while *task parallelization* has separate threads performing different operations on different items of data [13].

Multiple Threads & Processes: When applying data parallelization it is possible to split data sets *symmetrically* (or equally) amongst allocated processes a priori. When employing task parallelization, on the other hand, *asymmetrical* data sets can be *atomized* and put into a queue for dynamic process retrieval. Since we worked with symmetrical data sets, we were able to omit atomization. In our approach, the size of the total data set cannot exceed the size of memory. We split the data set across processor cores of a single machine. Within a distributed computing network (or big data architecture) data records should be split amongst available *worker nodes* [14]. There is a key difference between multiple threads and multiple processes: Processes do not share memory while threads share both state and memory. When performing multithreading, code segments are scheduled by the Operating System (OS). In the presence of only one core (or process) the OS creates the illusion of running multiple threads in parallel when in fact, it switches between the threads quickly, which is referred to as time division multiplexing [10].

Global Interpreter Lock (GIL): Multithreading in Python is managed by the host operating system and uses a mechanism called the Global Interpreter Lock (GIL), which limits the number of running threads to exactly one at a time. The GIL implicitly guards data structures from concurrent and possibly conflicting (write) access to avoid race conditions. Python prefers a serial development approach and takes a trade-off between *easy scripting* and *code performance*. It needs to be noted that the GIL, depending on the Python implementation, is temporarily released for Input Output (I/O) operations or every 100 bytecode instructions. This is the case of CPython implementations, which is the default on most systems. IronPython, PyPy and Jython "do not prevent running multiple threads simultaneous on multiple processor cores"[10, p. 4]. In order to get around CPython's GIL it is necessary to spawn new processes. Hardware specifically speaking, two different multiprocessor architectures exist [15]: 1) Symmetric (equally strong) multi-core chips; 2) Asymmetric multi-core chips with varying processing power.

Python offers a number of packages for spawning processes. However, a particularly useful one is `multiprocessing`. Bundled with it come queues. Only through such package implementations [16] is it possible to take advantage of today's multi-core chip architectures with Python.

III. Implementation & Results

Figure 2 summarizes writing speed results. It becomes clear that a scalable application needs multiprocessing capabilities. It would be best for concurrent insert processes to dispatch dynamically in high load scenarios. High frequency input should be queued, distributed and written as batch operations within intervals.

Single Thread Synchronous Inserts: Inserts into Cassandra take about 1.6 ms on average. In order to compare execution

speeds 1,000 and 100,000 records are inserted. Insert operations are executed four times and a simple arithmetic mean of the resulting run times is calculated. In other words, the mean runtime of 1,000 and 100,000 sequential synchronous inserts is 0.9 seconds and 80 seconds, respectively.

Single Thread Concurrent Inserts: Insert speed of large datasets can further be improved via concurrent writes through the Python Cassandra driver. This package exploits Cassandra's Staged Event-Driven Architecture (SEDA) [17]. We went with the recommended setting of core connections multiplied by one hundred `concurrency = 100` [18]. The Cassandra driver invokes a multiprocess instance for concurrently writing while the main program is still under the GIL. Following the same measurements approach, the arithmetic mean of run times equals 0.27 seconds and 29.5 seconds respectively. In essence, this result trisects the synchronous single thread performance.

Multiprocess Concurrent Inserts: Through leveraging Python's multiprocessing module with the Cassandra driver and its SEDA abilities, as introduced in Section II, writing speed can further be accelerated. Following the idea of data parallelization introduced in Section II the workload is distributed onto a number of separate processes as depicted in Figure 3. While all processes run simultaneously, each process is executed on a separate core, the dataset is split equally and distributed accordingly. Execution of 1,000 inserts in parallel yields a meager 0.03 seconds improvement, while 100,000 inserts distributed amongst four processes take 11.2 seconds in total. Each trial's total execution time was determined by the slowest process' runtime.

Cassandra Query Language (CQL): When testing the hypothesis from Section II regarding topic related word convergence tweets need to be retrieved from the database. Cassandra does not support the Structured Query Language (SQL) standard with respect to `GROUP BY`, `ORDER BY` clauses. Cassandra uses `ORDER BY` implicitly through table declaration via `WITH CLUSTERING` on primary key columns [19]. The order of the composite primary key declaration matters. The partition key decides on which physical node data are eventually stored by mapping rows. The cluster key decides the order of given rows [20]. When dealing with time series data it is advisable to create sectional time intervals. When attempting to retrieve most recent values the clause `WITH CLUSTERING ORDER BY (timestamp DESCENDING)` is needed for table declaration. This overrides the default ascending order [20]. Cassandra does not allow `JOIN` operations. Filtering is supported but may cause malfunctions due to the nature of Sorted String Tables (SSTables) and Log-Structured Merge-Trees (LSM-Trees) [21], [22]. We query for relevant values, allow filtering and iterate returned values via Python lists. The following constitutes a valid example: `SELECT column_a FROM keyspace.table WHERE column_b > 0 ALLOW FILTERING;` [19].

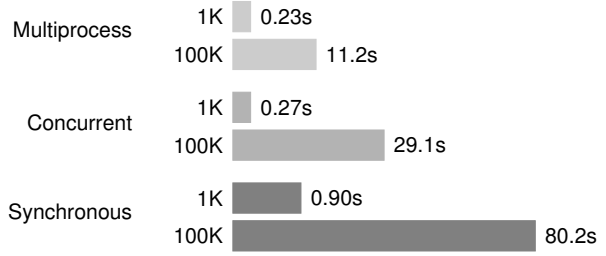

Figure 2. Comparing elapsed seconds inserting records into Cassandra

Tombstones are deletion markers [17]. It needs to be pointed out that they also occur when inserting *null values* or *collections*. Time To Live (TTL) data expiration may be an additional cause for Tombstone creations [23]. If a query exceeds the default value of 1,000 tombstones with respect to the `tombstone_warn_threshold` Cassandra issues and logs a warning. When, however, the `tombstone_failure_threshold` exceeds 10,000 tombstones Cassandra aborts the query.

When attempting to isolate tombstone issues it may be worth executing `sstabledump` in the Bourne Again Shell (`BASH`). This produces a JavaScript Object Notation (JSON) file of Cassandra's SSTable, which shows deletion markers as `"d"` next to data entries [23]. However, with Cassandra active, it is recommended to run `flush` first. This *flushes* all in memory data structures (MemTables) of associated tables to disk.

Public Debate Example: An exemplary headline by *The Hill* with `status_id = 974246607607779328` recorded 656 replies. This amount of replies is deemed sufficiently large for demonstrative purposes regarding topic related word convergence. "Senate [Grand Old Party (or Republican Party)] GOP: We will grow our majority in midterms" [24] is the headline of the chosen debate. Preliminary results indicate "that the general opinion, regarding the given headline, appears to be ridicule. Overall, little credence is given [... to] the Trump administration [25]".

Once noun-phrases and their respective user's follower counts have been retrieved from Cassandra and appended to a Python list they need to be traversed. The goal of this operation is to convert the list into a more tractable data structure. Dictionaries, which hold key value pairs, are great for this purpose, as the number of followers can be preserved, which allows more versatile analyses.

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

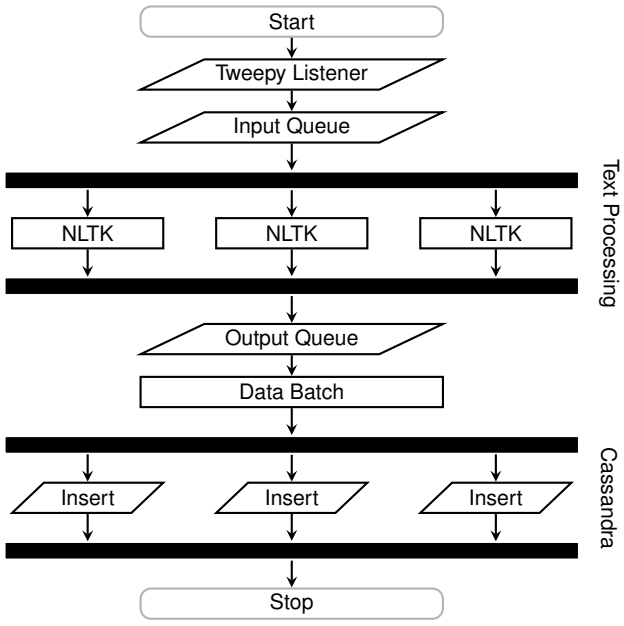

Figure 3. Concurrent Python multithread architecture to bypass GIL

dictionary entries to the Python console. Naturally, the number of followers indicates the *influential weight* of a post whereas overall noun frequencies yield the *public consensus*. Lower case converted stemmed word frequencies best illustrate topic related word convergence. This is due to the fact that the effects of word differences, capitalization, misspellings and affixes are effectively minimized.

IV. Conclusion & Discussion

Elaborating and realizing a scalable data-intensive text processing application relies on several components: 1) suitable data sources, 2) associated use cases, 3) processing power and memory size, 4) data persisting methods. Twitter is one of the biggest social platforms and provides a myriad of data sets. The choice of Cassandra as a highly available database is advocated and its inner workings are illuminated. Overall, a scalable text processing application is developed. The development of a big data application is taken on holistically and finds both innovative and uncommon solutions. Streaming Twitter data in the context of text mining and persisting it with Cassandra has not been discussed as such in literature. Twitter's *discussion* related topics, noun-phrases and word frequency convergences are both reviewed and analyzed in a novel way. This paper specifically contributes to a means of analyzing public debates (or discussions) in (near) real-time. Simultaneously, data is persisted without superimposing topic restrictions via search terms. Introduced methods could also be employed for public surveillance.

The introduced Python prototype still needs better functional `multiprocessing` package integration. Overall, three independent process groups should run concurrently and use queues to communicate as depicted in Figure 3. Our results indicate that inserts into Cassandra can be greatly improved through data parallelization. Future research should focus on accruing records in memory before batch insertion.

Conclusively, it is recommended to keep the real-time streaming focus a priority. Cassandra proves to be an exquisite choice for both data persistance and retrieval. Last but not least, while this paper focused rather on the processing and storage performance of our architecture, further optimization towards an adequate retrieval and re-analysis approach seem promising by combining Cassandra with Apache Hadoop Distributed File System (HDFS) [26] and exploiting Apache Spark [27] on top of an HDFS-based data lake.

Acknowledgment

This work is based on the thesis of G-P. Heine named *'Developing a Scalable Data-Intensive Text Processing Application with Python and Cassandra'*, University of Applied Sciences Wiener Neustadt, April 2018.

References

[1] D. Laney, "3d data management: Controlling data volume, velocity and variety," META Group Research Note, vol. 6, no. 70, 2001.

[2] P. Zikopoulos, C. Eaton, and IBM, Understanding big data: Analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media, 2011.

[3] D. e. a. Jiang, "epic: an extensible and scalable system for processing big data," The VLDB Journal, vol. 25, no. 1, 2016, pp. 3–26.

[4] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," Information Sciences, vol. 275, 2014, pp. 314–347.

[5] I. Feinerer, D. Meyer, and K. Hornik, "Text mining infrastructure in r," Journal of Statistical Software, vol. 25, no. 5, 2008, pp. 1–54.

[6] R. Dale, "Classical approaches to natural language processing," in Handbook of Natural Language Processing, Second Edition, N. Indurkhya and F. J. Damerau, Eds. Chapman and Hall/CRC, 2010, vol. 2, ch. 1, pp. 3–7.

[7] B. Liu, Sentiment Analysis. Cambridge: Cambridge University Press, 2015. [Online]. Available: ebooks.cambridge.org/ref/id/CBO9781139084789

[8] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, "Predicting elections with twitter: What 140 characters reveal about political sentiment." vol. 10, no. 1, 2010, pp. 178–185.

[9] B. O'Connor, R. Balasubramanyan, B. R. Routledge, and N. A. Smith, "From tweets to polls: Linking text sentiment to public opinion time series." vol. 11, no. 122-129, 2010, pp. 1–2.

[10] N. Singh, L.-M. Browne, and R. Butler, "Parallel astronomical data processing with python: Recipes for multicore machines," Astronomy and Computing, vol. 2, 2013, pp. 1–15.

[11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in Proceedings of the April 18-20, 1967, spring joint computer conference. ACM, 1967, pp. 483–485.

[12] S. Binet, P. Calafiura, S. Snyder, W. Wiedenmann, and F. Winklmeier, "Harnessing multicores: Strategies and implementations in atlas," in Journal of Physics: Conference Series, vol. 219, no. 4. IOP Publishing, 2010, pp. 1–7.

[13] D. Gove, Multicore Application Programming: For Windows, Linux, and Oracle Solaris. Addison-Wesley Professional, 2010.

[14] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning spark: lightning-fast big data analysis. "O'Reilly Media, Inc", 2015.

[15] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," Computer, vol. 41, no. 7, 2008, pp. 33–38.

[16] Python Software Foundation, "Python/c api reference manual," *https://docs.python.org/2/c-api/*, 2018, retrieved: 09, 2018.

[17] H. Eben, Cassandra: The definitive Guide. O'Reilly Media, Inc., 2010.

[18] DataStax, "Python cassandra driver," *datastax.github.io/python-driver/index.html*, 2017, retrieved: 02, 2018.

[19] Apache Software Foundation, "Cassandra query language (cql) v3.3.1," *cassandra.apache.org/doc/old/CQL-2.2.html*, 2017, retrieved: 03, 2018.

[20] P. McFadin, "Getting started with cassandra time series data modeling," *patrickmcfadin.com/2014/02/05/getting-started-with-time-series-data-modeling/*, 2014, retrieved: 03, 2018.

[21] G. Graefe, "Modern b-tree techniques," Foundations and Trends in Databases, vol. 3, no. 4, 2011, pp. 203–402.

[22] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, Inc., 2017.

[23] A. Babkina, "Common problems with cassandra tombstones," *opencredo.com/cassandra-tombstones-common-issues/*, 2016, retrieved: 04, 2018.

[24] A. Bolton, "Senate gop: We will grow our majority in midterms," *thehill.com/homenews/senate/378517-senate-gop-we-will-grow-our-majority-in-midterms*, 2018, retrieved: 03, 2018.

[25] G.-P. Heine, "Developing a scalable data-intensive text processing application with python and cassandra," Master's thesis, University of Applied Sciences Wiener Neustadt, 2018.

[26] Apache Software Foundation, "Welcome to Apache Hadoop!" *http://hadoop.apache.org/*, 2014, retrieved: 09, 2018.

[27] Apache Software foundation, "Apache spark unified analytics engine for big data," *http://spark.apache.org/*, 2018, retrieved: 09, 2018.