Seminar Thesis: Applying Fuzzy Logic to Data Warehouses

Fuzzy Concepts on Real Time Applications

Steve ASCHWANDEN
Stefan NÜESCH

Examiner
Dr. Daniel Fasel

Wabern, December 2, 2013

Contents

1 Introduction

The number of new tweets per second on the social media platform Twitter[1] is huge (over hundred thousand per second). To handle such a big amount of data, a scalable, fault-tolerant real-time framework has to be used. Storm was benchmarked at processing one million 100 byte messages per second per node[2]. It allows to classify every new tweet, for example with a fuzzy logic approach. The so called "Spout" is responsible for fetching the tweets and the "Bolts" can do some transformation on the received data or persist them in some sort of storage. Cassandra[3] is an ideal solution for the given scenario, because it is a distributed, elastically scalable, highly available, fault-tolerant and column-oriented database.

2 Problem Statement

2.1 Research Questions

- How can fuzzy classifications be applied to twitter feeds using storm?
- How can the results of such classifications be stored in a cassandra column store?

3 Objectives

The main objective of this seminar project is to develop a simple prototype, which uses storm to fuzzily classify messages from twitter feeds and store the results in a Cassandra NoSQL database. The resulting application will use Storm on local system only (not on a cluster). Furthermore, only one fuzzy concept will be implemented in combination with a simple semantic analysis of the twitter feeds. However, the prototype shall be easily extensible with more complex analytics and further fuzzy concepts.

4 Methodology

This thesis will be conducted by prototyping.

4.1 Prototype

The prototype resulting from this project will be a Java application based on the Storm library and using Cassandra for persistence. For analysing the twitter feeds, SentiWordNet will be used.

5 Existing Research

5.1 Twitter & Storm

5.2 Storm with Cassandra

5.3 SentiWordNet

6 Technologies

6.1 Storm

Storm is an open source distributed real-time computation system and programming language independent with the following nice properties:

[1] https://www.twitter.com
[2] http://storm-project.net
[3] http://cassandra.apache.org

- **Easy to program** An implementation of a real-time processing tool from scratch needs a lot of time and effort. With storm, a simple working topology can be created within a few hours
- **Programming language independent** The easiest way is to develop in a JVM-based language, but Storm support any language if you use or implement a small intermediate library
- **Fault-tolerant** If a worker is going down, the cluster will reassign tasks when necessary
- **Scalable** Add or remove machines to the cluster. Storm will reassign tasks to new machines as the are available or distribute the tasks from the removed machines to the others which are still there
- **Reliable** Storm guarantees that every message will processed at least once. If an error occurs, the messages can be processed more then once. But a message will never be lost
- **Fast** Storm can process one million 100 byte per second per node on a machine with two processors (Intel E5645@2.4Ghz) and a 24 GB memory

6.1.1 Components

A storm topology consists of two main components:

- Spouts are responsible to fetch the data from a source (e.g. text file, twitter streaming API)
- Bolts are responsible for the transformations of the fetched data

The spout acts as the source of the data flow and the last bolt is the sink which store the output to a database. The output of one component goes always to the input of the next, therefore no data will flow through an already visited node. A simple example should help to illustrate the components of the storm architecture (Figure 1):

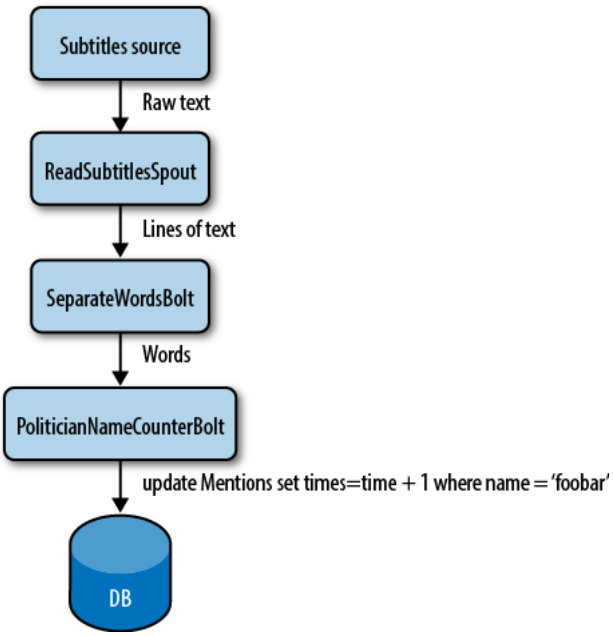


Figure 1: Storm example

All the spoken words during the TV news show will be stored in a text file. If a new sentence was written into the file the spout hands them to the responsible bolt which separate the sentence

into words. This stream of words is passed to another bolt that compares each word to a list of politician's names. If there is a match, the second bolt increases the counter for that name in a storage. To check the number of occurrence, query the database which is updated in real-time.
Typical uses cases for a storm application can be:

- **Processing streams** This is the case in shown example
- **Continuous computation** Send continuously data to the clients so they can update and display results in real-time, for example as site metrics
- **Distributed remote procedure call** Easily parallelize CPU-intensive operations

6.1.2 Grouping

After the implementation of the spouts and bolts for the topology, they have to be connected with each other. Stream grouping defines which stream are consumed by each bolt and how they will be consumed. Strom provides different grouping types:

- **Shuffle** The most commonly used grouping. It sends each tuple emitted by the source to a randomly chosen bolt. If there are no dependencies between the emitted tuples (e.g. fuzzy classification of a tweet) this grouping method is used
- **Fields** Control how the tuples will be distributed, based on the emitted fields. It guarantees that a set of values for the same combination of fields will be sent to the same bolt. This is useful to implement a MapReduce functionality
- **All** Sends a single copy of each tuple to every receiving bolt (e.g. resetting a counter, refreshing a cache)
- **Custom** Create a custom stream grouping by implementing the *back.type.storm.grouping.CustomStreamGrouping* interface

6.1.3 Technical Details

In a storm cluster there are 2 node types, a master node and worker nodes. On the master node runs a daemon called 'Nimbus', which is responsible for code distribution, assigning tasks to the worker nodes and failure monitoring. A portion of the topology is running on the worker nodes (responsible daemon is called 'Supervisor'). A topology runs across multiple worker nodes on different machines. Since Storm keeps the states of the cluster either in 'Zookeeper' or on local disk, the daemons are stateless and can fail or restart without disturbing the system.
Storm uses an advanced, embeddable networking library zeromq (0mq, http://zeromq.org/), which provides features that make Storm possible. Some characteristics of zeromq:

- Act as a concurrency framework
- Faster than TCP, ideal for clustered products and supercomputers
- Carries messages across in-process, IPC (inter-process communication), TCP and multicast
- Asynchronous I/O model for scalable multi-core applications
- Connect sockets N-to-N

Storm topologies can run in two different modes. The local mode is a single Java Virtual Machine on a local machine. This mode is used for developing, testing and debugging. The other possibility is to run the software on different remote machines. But there are no debugging information in the production mode.

The difference between a traditional BigData approach like Hadoop and storm is the paradigm that it addresses. Hadoop fundamentally a batch processing system. The results are available after the whole data was distributed over the whole system. The construction of topologies that

transform unterminated streams of data is supported by storm. A storm topology will never stop running (until termination by user), Hadoop jobs will stop after all available data (at starting time) proceeded by the system.

6.2 Cassandra

Eben Hewitt explained Cassandra with only 50 words [7]:

"Apache Cassandra is an open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, column-oriented database that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable. Created at Facebook, it is now used at some of the most popular sites on the Web."

The data store is an open source Apache project available at http://cassandra.apache.org. Cassandra originated at Facebook in 2007. It should solve the company's inbox search problem, because they had to deal with large volumes of data. The project was moved to Apache in March 2009. Cassandra has some interesting properties:

6.2.1 Elastic Scalability

There are two different types of scaling. Add more hardware capacity and memory to the existing way is one way to handle a greater number of requests. This approach is called vertical scaling. Horizontal scaling will add more machines to distributed the incoming data over multiple nodes. This will decrease the amount of data for every machine. Elastic scalability is a specialization of the horizontal scaling where the cluster can scale up and scale back down, depending on the current load. Cassandra is able to handle new machines on-the-fly, no restart of the process or manually rebalancing of the data is required.

6.2.2 High Availability and Fault Tolerance

Machines in the cluster will have technical issues from time to time. Cassandra is able to handle replaced failed nodes without downtime. Also replication of data to multiple data center is possible to prevent downtime if a data center experiences a catastrophe like a fire.

6.2.3 Tuneable Consistency

Consistency means that a read operation returns the last written value from the database. But a database system cannot implement all three components of the CAP theorem and the focus of Cassandra are Availability and Partition tolerance (AP). Therefore, the consistency is not guaranteed. But it allows to configure the degree of the consistency in balance with the level of availability:

- **Strict consistency** Also called sequential consistency. It's the highest level of consistency and it requires that a read will always return the recently written value in the database. If you have only a single machine with one processor (and one processor clock) the last written value is easy to find. But if there are geographically dispersed data centers, it becomes much more slippery. A global clock which can timestamp all operations, regardless of the data location or how many services are needed to determine the response, is required
- **Causal consistency** A weaker form of the strict approach. It uses a much more semantic approach and it does away with the fantasy of a single clock which can be synchronized without creating a bottleneck. The idea is that writes which are potentially related must be read in sequence. Writes are inferred not to be causally related, if two operations suddenly write to the same filed (operations must be different and unrelated). But if there is enough time between two write operations, we assume that they are causally related. Therefore, causal consistency says that causal writes must be read in sequence

- **Weak (eventual) consistency** All updates will propagate throughout all of the machines in a distributed system. Of course, this will take some time. But It could be possible that all replicas will be consistent

The user of Cassandra can control the number of replicas to block on for all updates (Tunable consistency). This can be done by setting the level of consistency against the replication factor. With the replication factor the client can define the level of consistency, but a higher level implies a lower performance of the system. The factor controls the number of nodes in the cluster where you want to distribute the updates (add, update and delete). The number of replicas which have to acknowledge a write operation or respond to a read operation to declare this operation as successful is defined by the consistency level.

6.2.4 Raw-Oriented

Cassandra is a column-oriented database, it represents it's data structures in sparse multidimensional hashtables. For every given row you can have many columns, but they doesn't have to be the all the same like in the relational model. A unique key makes the data accessible. One advantage of Cassandra is that you don't have to define what the data structure must look like or what fields you will need. This implies also that you can add or remove new fields on the fly without disturbing the system. This is ideal if you use an agile development methodology and you don't have time to define the entire database schema at the beginning of the project.

6.2.5 Schema-Free

The first step to define a Cassandra database is the creation of a keyspace which is a logical namespace to hold column families and some configuration properties.

6.2.6 High Performance

Cassandra was designed to take full advantage of multiprocessor/multicore machines and run on multiple of these machines in data centers. It should be scale to hundreds of terabytes and stay consistent. People from the university of Toronto analysed the performance of Cassandra [8]. They defined some different workloads to create some scenarios. One was with 50% inserts, 25% reads and 25% scans. Two independent clusters are used where each has about 16 Linux nodes (Intel Xeon quad core CPU's, 16GB of RAM, 148GB disk space). The nodes are connected with a gigabit Ethernet network over a single switch. The strength of Cassandra is clearly visible (Figure 2), because the throughput with only one node is lower than other ones. If the number of nodes was increased, the gap between Cassandra and all the other tested systems is huge.

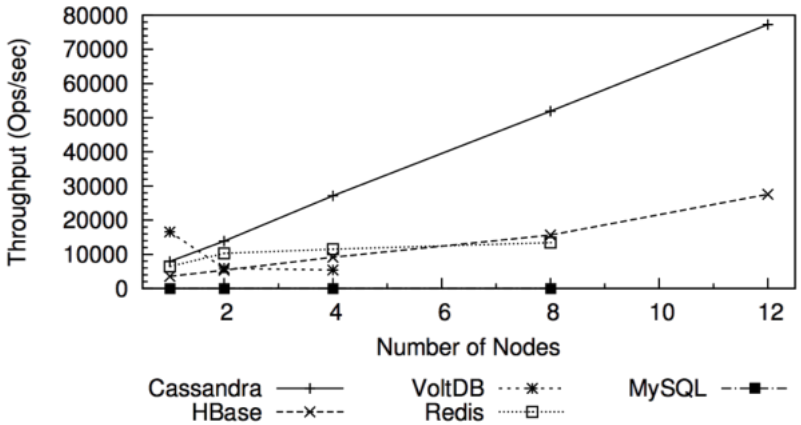


Figure 2: Performance of different distributed database systems

The authors of the paper concluded their paper with the following words:

"In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments with a linear increasing throughput from 1 to 12 nodes."

7 Implementation

7.1 Architecture

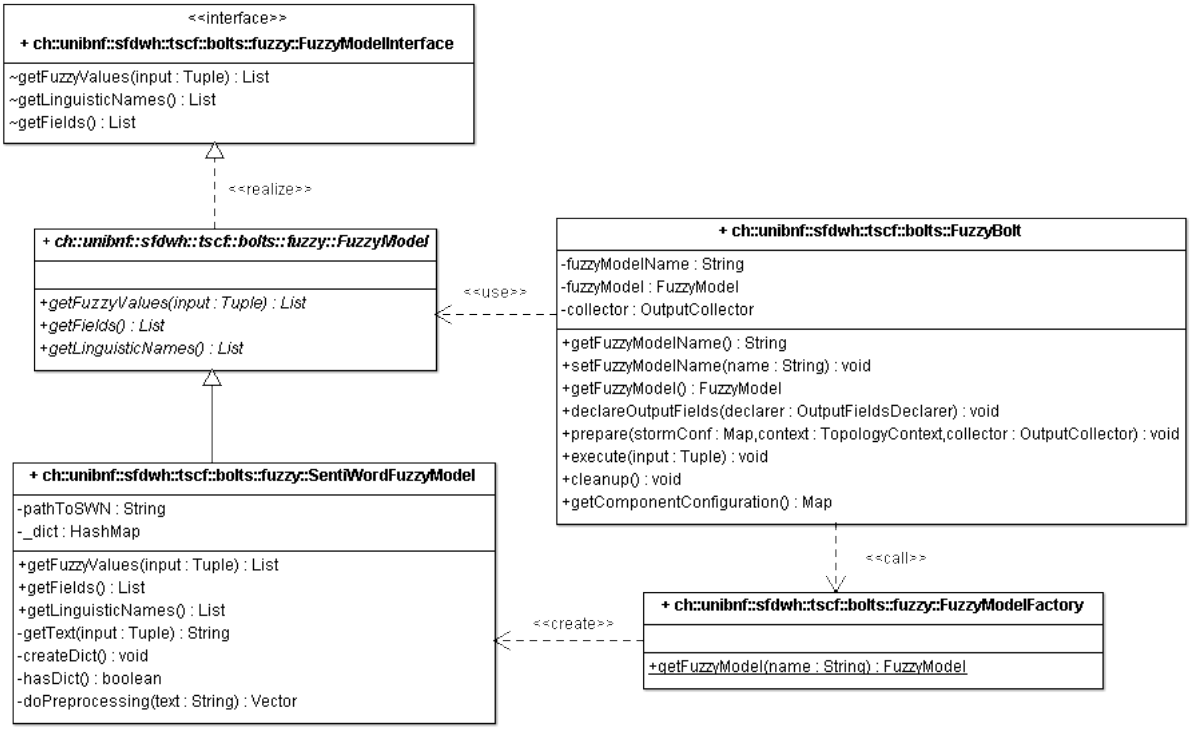
7.2 SentiWordNet Analysis

SentiNetWord is an English dictionary for opinion mining (OM, also known as sentiment classification). The goal was to decide for every word of the WordNet if it has a positive, negative or neutral meaning. WordNet is a large lexical database of nouns, verbs, adjectives and adverbs provided by the Princeton University. The developers calculated for every word three values between 0.0 and 1.0 and they had to sum up to one. They labelled different entries of the dictionary manually to define a ground truth for the training set. Then a set of classifiers was trained with the provided data. After these steps the classifiers were able to calculate the corresponding values of the remaining words which weren't in the training data samples. Since the first version of SentiWordNet in 2006 a lot of improvements are done, the actual version is 3.0.0. It's freely available for research approaches.
The dictionary consists of 117'659 entries (69.8% nouns, 15.4% adjectives, 11.7% verbs and 3.1% adverbs). If we take a closer look at the calculated values (in SentiNetWord 1.0.0) we can see that most of the words are objective (about eighty percent are greater than 0.75). Only a few words are positive (0.48% $>$ 0.75; about 560 words) or negative (1.28% $>$ 0.75; about 1500 words).
The resulting dictionary can be downloaded on the research website of SentiWordNet and has the following columns (tab separated):

1. Part of speech (noun, verb, adjective, adverb)
2. ID
3. Positive score
4. Negative score
5. The words (plural because of synonyms)
6. Glossary (the word in a given context)

The neutral (objective) score of the word can be calculated by 1 - (Positive score + Negative score).

7.3 Fuzzy Concept Implementation



To delegate the creation of the model with the implemented fuzzy approach to the *FuzzyBolt* class, a Parameterized Factory Method pattern is used. It's possible to create a new fuzzy model approach by extending the *FuzzyModel* class and modify the *FuzzyModelFactory* to make the object creation of the corresponding model accessible. The selection of the new approach can be done by the following statements:

```
FuzzyBolt fuzzyBolt = new FuzzyBolt();
fuzzyBolt.setFuzzyModelName(``our approach'');
```

The second line will call the static *getFuzzyModel* method of the factory. The *SentiWordFuzzyModel* class has a *createDict* method which creates a hash map out of the text file consists of the word as key and the corresponding vector with the opinions (positive, negative and neutral) as value. The *FuzzyBolt* class is the first bolt in the whole Storm process and receives the output of the twitter spout. The execute-method calculates the three sentiment values of the tweet and give them to the Cassandra bolt.

8 Conclusion

9 References

References

[1] Andrea Esuli, Fabrizio Sebastiani *SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining* available: `http://sentiwordnet.isti.cnr.it/`, accessed 18th November 2013.

[2] Stefano Baccianella, Andrea Esuli, Fabrizio Sebastiani *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining* available: `http://sentiwordnet.isti.cnr.it/`, accessed 18th November 2013.

[3] Alec Go, Lei Huang, Richa Bhayani *Twitter Sentiment Analysis* available: `http://www-nlp.stanford.edu/courses/cs224n/2009/fp/3.pdf`, accessed 18th November 2013.

[4] M. Tim Jones *Process real-time big data with Twitter Storm* available: `http://www.ibm.com/developerworks/library/os-twitterstorm/os-twitterstorm-pdf.pdf`, accessed 18th November 2013.

[5] Shalom N. *Nati Shalom's Blog: Real-Time Big Data with Storm, Cassandra, and XAP; April 4th, 2013* available: `http://natishalom.typepad.com/nati_shaloms_blog/2013/04/real-time-big-data-with-storm-cassandra-and-xap-1.html`, accessed 18th November 2013.

[6] Jonathan Leibiusky, Gabriel Eisbruch and Dario Simonassi *Getting Started with Storm;* 30th August 2012; O'Reilly, 978-1-449-32401-8

[7] Eben Hewitt *Cassandra - The Definitive Guide;* November 2010; O'Reilly, 978-1-449-39041-9

[8] Tilmann Rabl, Mohammad Sadoghi, Hans-Arno Jacobsen, Sergio Gomez-Villamor, Victor Muntes-Mulero, Serge Mankovskii *Solving Big Data Challenges for Enterprise Application Performance Management* available: `http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf`, accessed 2nd December 2013.