Near Real-time Detection of Crisis Situations

Sylva Girtelschmid*, Andrea Salfinger†, Birgit Pröll‡, Werner Retschitzegger§, Wieland Schwinger¶
*‡ Inst. for Application Oriented Knowledge Processing, †§¶ Dept. of Cooperative Information Systems
Johannes Kepler University Linz
Altenbergerstr. 69, 4040 Linz, Austria
K0956640@students.jku.at*, {andrea.salfinger†, werner.retschitzegger§, wieland.schwinger¶}@cis.jku.at, bproell@faw.jku.at‡

***Abstract*—When disaster strikes, be it natural or man-made, the immediacy of notifying emergency professionals is critical to be able to best initiate a helping response. As has social media become ubiquitous in the recent years, so have affected citizens become fast reporters of an incident. However, wanting to exploit such 'citizen sensors' for identifying a crisis situation comes at a price of having to sort, in near real-time, through vast amounts of mostly unrelated, and highly unstructured information exchanged among individuals around the world. Identifying bursts in conversations can, however, decrease the burden by pinpointing an event of potential interest. Still, the vastness of information keeps the computational requirements for such procedures, even if optimized, too high for a non-distributed approach. This is where currently emerging, real-time focused distributed processing systems may excel. This paper elaborates on the possible practices, caveats, and recommendations for engineering a cloud-centric application on one such system. We used the distributed real-time computation system Apache Storm and its Trident API in conjunction with detecting crisis situations by identifying bursts in a streamed Twitter communication. We contribute a system architecture for the suggested application, and a high level description of its components' implementation.**

I. Introduction

During the last decade, social media channels have evolved to the point of becoming omnipresent in our everyday lives. Platforms, such a Twitter, play a vital role in sharing information fast. It should come to no surprise that the Twitter data pool is popular for information mining as it can reveal valuable insights, be it for crisis monitoring applications, or marketing, to monitor the perception of a new product. Moreover, being able to utilize Twitter data to promptly detect a situation that requires a responsive action from a rescue crew, e.g. during natural disasters, can often save lives. However, even the situation detection alone is non-trivial. For a human, processing Twitter conversations for this purpose would mean having to look into the content of each message and extract the interesting information from it. A more efficient approach, better suited for a computer, is to cluster similar posts without having to first consider the semantics of the content. To satisfy the real-time detection demands, techniques for bursty topics identification are applicable here.

Bursts are in this context collections of posts mentioning the same topic more often within the current period than within the previous one. The problem of detecting burst topics falls under a broader category of research dealing with Topic Detection and Tracking (TDT) that has been extensively addressed in academic articles.

Of the various TDT approaches, clustering is a popular one to identify bursts. Other methods for burst detection include finite state automata, Fourier transform, time series, or Wavelet transform [1]–[3]. For our first setup, the clustering approach appealed the most to us thanks to its inherent property of being readily parallelizeable.

However, the enormous throughput of the real-world streams of Twitter posts prohibits timely online processing of a sequential implementation of state-of-the-art methods for topic clustering. For an acceptable performance, this problem requires single-pass, and optimized methods, as well as scalable implementation. In the recent past there has been work on successfully applying Cloud technologies to the online clustering problem to guarantee scalability, and near real-time processing of streaming data [4]–[6]. In this paper we show that Cloud technologies have a potential in improving responsiveness during emergency situations through mining social media content.

Since our system also needs to work with persistent data, we decided to make use of the Trident API, a high-level abstraction of Apache Storm, as it makes stateful processing more manageable than Storm alone. Although Trident is gaining popularity, to the best of our knowledge, we are not aware of any other systems that employ Trident for the use of Burst Topic Detection. In our work, we evaluate Trident's applicability to detecting an outburst of a crisis situation in near real-time. We propose an architecture for such an online disaster detector system and contribute a high level description of a Trident topology implementation.

The rest of this paper is organized as follows: In the next section we discuss a number of research areas related to our work and identify the specific ideas from which our devised system borrows. In sec. III, we detail our approach to detecting new emergency situations from Twitter data streams. Sec. IV discusses our test cases. Finally, in sec. V, we conclude on our findings, and provide an outlook on future work.

II. Related Work

Due to the multi-disciplinarity of the envisioned application domain, related approaches, and valuable preparatory work need to be drawn from several areas. Techniques essential to address the requirements imposed by the crisis management application domain can be found in the research fields of Event Detection, First Story Detection, Burst Detection, Knowledge

Bases, Keyword and Topic Extraction, and Parallelization, which we will motivate and discuss in the following.

a) Event Detection: Finding a common topic within a set of documents has been covered extensively in research (TDT initiatives). In the subject of crisis monitoring, this research also finds its application, as event/story detection is in this subject's heart. There are various approaches to this problem: e.g. in [7], [8], the method of clustering documents based on word similarity using the Vector Space model (VSM) is presented. In [9], the clustering of the documents is improved by considering the locality information in the process of discriminating the events into the clusters. In [10], the similarity score calculation also incorporates the social network structure besides the content of the message. Yet another class of solutions for event detection, and tracking bases on probabilistic methods, such as presented in [11]–[13]. There, the actually encountered message density w.r.t. the specific topics is compared to an expected density. A valuable work summarizing research on event detection in the realm of Twitter can be found in [14].

b) Detection of Novel Events: Although organizing messages into clusters brings a useful insight to our work, it is not sufficient for our problem solution. We have to consider methods capable of identifying those topics that have not been discussed before within some long enough time period. In other words, we need to identify posts that are discussing a new story. First Story Detection (FSD) methods are therefore well applicable here, as they are designed to detect when a document discusses a previously unseen content. Examples of FSD implementation are presented in the works of [15], [16]. The authors propose an optimized approach to FSD, which makes it well suited for identifying events online, i.e. for identifying events from real-time streaming text such as tweets. The ideas set out in these works form the basis of our new event detection algorithm.

c) Identifying Bursts: Typically, whenever a newsworthy event occurs, many people tend to share information about it on social media. This causes a temporal burst of closely related messages which can be captured [1], [17]. Detecting bursts has also been well studied [1]–[3], [18]. For example, in [3], the author achieves real-time event detection by clustering wavelet-based signals. Wavelet transformation technique is applied to build signals for individual words, which are cross correlated and the signals are then clustered using a scalable eigenvalue algorithm. In our approach, we use a bucketing method, similar to that proposed by [15], [16], since it directly fits the parallel programming paradigm we adopted.

d) Knowledge-based Sensing: In our system, we also need to consider techniques that allow us to report only events which we are interested in. In this context, an event is defined as a topic that suddenly draws the attention of the public. As such, it may often be of no value for our purposes, since, for example, news about celebrity deaths are also causing abrupt increase in related posts being sent. Our application needs to be able to identify only those events that are related to disasters. To achieve this, a disaster ontology may be employed. There have been many attempts in building an ontology related to disaster. It is, however, often the case that existing disaster ontologies focus only on a specific type of disaster, which they explore and describe in detail [19], [20]. For our purposes, it is sufficient to have a high level disaster dictionary to identify the particular disaster situation referred to in the tweets. Working with multiple languages would, therefore, be also feasible.

e) Extracting Keywords: Document keyword and keyphrase extraction is another complex problem addressed in the research [21]. The frequent techniques include word frequency analysis, distance between words, or lexical chains to rank the keywords. When it comes to keyphrase extraction, graph-based ranking methods are being successfully used. However, such methods are proposed primarily for a single document or a document collection. This is not directly applicable when it comes to extracting representative words from such a short document as a tweet. The large variety of topics found in tweets also makes this task more challenging. In [22], the authors propose a method specifically for keyword extraction from Twitter. The approach is based on organizing keyphrases by topics learnt from Twitter. For our purposes, the best approach is to apply a content-sensitive keyword extraction, as our tweet collection from which we need to extract the keywords is already formed of similar content.

f) Meeting Real-time Demands: The algorithms that can achieve our task of detecting a new story from a real-time stream of data are comparatively computationally expensive. To safely achieve real-time response of a system processing the full streaming data flow from Twitter (the so called 'firehose'), parallelization of the algorithm is necessary. Some recent studies have shown that the implementation of clustering algorithms on the Storm distributed framework is reasonable [4], [5]. In our implementation, however, we utilize the higher level Storm API called Trident. This design benefits from the guarantee of exactly-once semantics[1], as well as the ability to process streaming messages in batches which performs better when querying a database system.

III. Implementation

Extracting useful information from Twitter data flow is no doubt a challenge. The posts are not only short (140 characters), and varied in topic but also highly noisy, in that most of the conversations contain a lot of useless babble. The task of identifying bursts offers itself as the best approach to finding information of value as more people will get drawn into a conversation about something vital than about a certain individual reporting on currently drinking the most delicious cacao.

In this project, we focus on utilizing one of the FSD technique which applies a Vector Space model and is optimized using Locality Sensitive Hashing. This technique is described in detail in [16], and we provide a brief overview

[1]During a node failure, only the messages that have not been fully processed will get resent. This is as opposed to Storm's at-least-once semantics, where some messages may get processed twice.

in subsection III-B1. This FSD algorithm outputs a similarity score for each incoming message together with the ID of the message to which it is the most similar. The next stage involves the actual clustering, or bucketing to identify bursts. This is performed based on the message's content similarity score and monitoring of the growth rate of the buckets in which similar tweets are gathered. The components of our system are described in sec. III-B.

A. Distributed Platform

Besides Spark Streaming[2], the Apache Storm Trident API is currently the best suited framework for our application. It simplifies the implementation of parallel tasks by providing a programming interface suitable for stream processing and continuous computation[3].

As already mentioned in sec. I, the successful use of Storm Apache in data stream clustering applications has recently been reported [4], [5]. The advantages of using the Trident API over the core Storm API in our application are threefold: First, Trident can guarantee exactly-once semantics as opposed to at-least-once semantics of Storm. This means, that we don't have to worry about already processed messages being resend in case of a failure in the computing cluster and therefore we will not be faced with potential erroneous bursty event reports.

Another advantage of using Trident is the fact that it handles streams of batches of tuples as opposed to streams of tuples. This achieves better performance for communication with a database.

Lastly, state persistence handling is incorporated generically in the Trident API which allows us to be flexible in the selection of our back-end technology. On the other hand, using Trident requires some additional checks for our algorithm, which we point out at the end of subsection III-B2.

B. System Components

Fig. 1 shows a component overview of the proposed system. In our setup, the topology is fed by data from a KafkaSpout, where the source of the stream is 1% of the Twitter firehose accessed through a Hosebird client[4].

The first component of our system implements the core parts of the open source project "First Story Detection on Twitter using Storm"[5] and adopts it for running on an actual distributed cluster with real-life data streams. For later queries, tweets are saved in the NoSQL distributed storage mechanism Apache Cassandra. The next component takes care of grouping of related tweets into buckets, which are monitored for bursts based on their growth rate. References to the bulk of tweets from the fastest growing bucket, i.e. the tweets form the burst, are also made persistent. The task of the third component is to decide whether the topic of the burst is of interest. The final component executes only if a crisis related new event was detected. Its task is to notify responsible operators from the area of the event occurrence, and to automatically spawn a new instance of a Twitter search tracking this event.

1) FSD using Locality Sensitive Hashing: In this section, we briefly explain the gist of the FSD component.[6] This component outputs additional information for each incoming message: the Twitter ID of the closest neighbour message (the most similar post), and a score of similarity in the range of $[0.0 - 1.0]$, where 1.0 identifies an identical post.

Every newly arriving message from the Twitter data stream is split into words and a number of preprocessing steps are applied (e.g. removal of URLs and mentions, replacement of some, often intentionally, misspelled words, as well as simplistic[7] word stemming). The cleaned up corpus is then submitted for calculation of the nearest neighbour.

First, this process involves determining the TF-IDF (Term-Frequency - Inverse Document Frequency) weighting for each term in the message to convert the representation of the tweet message into a vector normalized by Euclidean norm. The new vector must then be compared to the vectors of the preceding messages. An approximate near-neighbour among the seen documents can be found fast using the Locality Sensitive Hashing algorithm. It works on the assumption that similar tweets tent to hash to the same value. This allows for an efficient optimization where the number of documents to which comparison has to be made is greatly reduced. Namely, it uses hash tables to bucket similar tweets so that each incoming tweet will be compared with only the tweets that have the same hash.Finally, cosine similarly measure is applied to compute the distance of the nearest neighbour from the incoming tweet. If the calculated distance to the closest tweet is below a predefined threshold, this algorithm also uses an additional step of comparing the distance to a fixed number of latest tweets. This step alleviates the problem of possibly overlooking a closer tweet posted in the immediate time proximity. Such problem can easily arise given the method's randomness of selecting tweets for comparison in the first place.

2) Bucketing and Identifying bursts: In effect, the process of grouping the posts that are similar also identifies a new event. In other words, if an incoming message was found to have a low similarity score, this message will be marked as a potential new event (by placing it in a new bucket). Then, given that there will be enough related/similar posts following it, this event will grow in its own cluster. In [16], this process is referred to as threading, whereas in [10] it is called the cluster summary. We chose to describe this process in two subtasks - one of bucketing and the other of identification of the bursts.

The first task of this component is to gather similar tweets in the same bucket. Before bucketing, tweets whose similarity score is unfavorable (too low) are filtered out. I.e. if $(1 - cosineDistance) < threshold$ the tweet is included in

[2]http://spark.apache.org/streaming/
[3]http://storm.apache.org/
[4]A robust Java HTTP library for consuming Twitter's Streaming API
[5]https://github.com/mvogiatzis/first-stories-twitter

[6]For detailed explanation, refer to the original author's website at http://micvog.com/2013/09/08/storm-first-story-detection/
[7]For performance reasons, we only apply stemming to English words that have a common stem in all their inflected variants

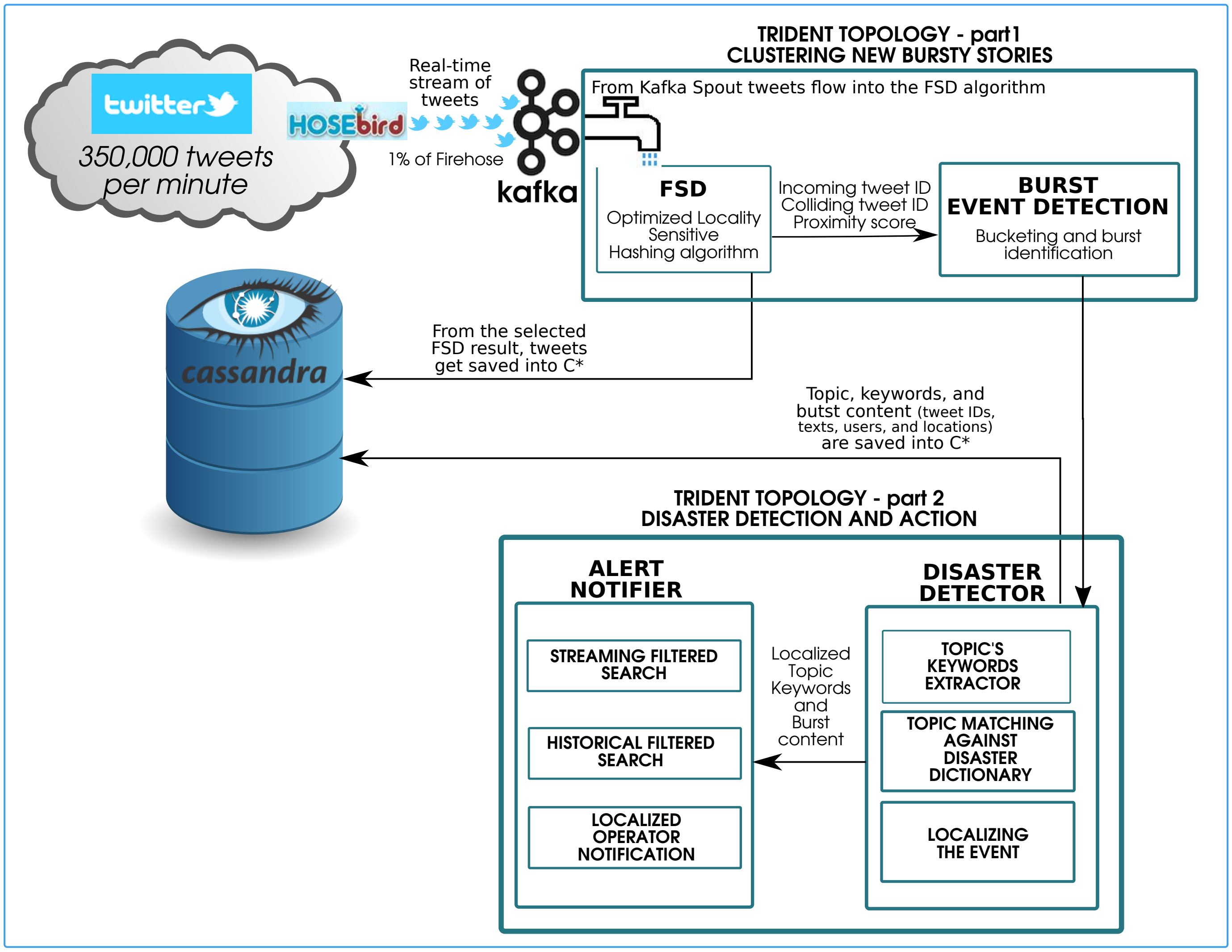
twitter
350,000 tweets per minute
HOSEbird
Real-time stream of tweets
1% of Firehose
kafka
TRIDENT TOPOLOGY - part1
CLUSTERING NEW BURSTY STORIES
From Kafka Spout tweets flow into the FSD algorithm
FSD
Optimized Locality Sensitive Hashing algorithm
Incoming tweet ID
Colliding tweet ID
Proximity score
BURST EVENT DETECTION
Bucketing and burst identification
cassandra
From the selected FSD result, tweets get saved into C*
Topic, keywords, and butst content (tweet IDs, texts, users, and locations) are saved into C*
TRIDENT TOPOLOGY - part 2
DISASTER DETECTION AND ACTION
ALERT NOTIFIER
STREAMING FILTERED SEARCH
HISTORICAL FILTERED SEARCH
LOCALIZED OPERATOR NOTIFICATION
Localized Topic Keywords and Burst content
DISASTER DETECTOR
TOPIC'S KEYWORDS EXTRACTOR
TOPIC MATCHING AGAINST DISASTER DICTIONARY
LOCALIZING THE EVENT

Figure 1: System architecture overview

further processing. The predefined threshold was found to give the best result if kept in the range of $[0.5, 0.6]$. As explained in [16], higher threshold values cause the topics in a bucket to be diverse and plentiful, while lower threshold makes the topics very specific and scarce. A passing tweet will then be placed either in a new bucket, if its nearest neighbour is not already in one of the existing buckets, or it will join the already existing cluster of similar tweets in their bucket. If it ends up in a new bucket, it is considered to potentially contribute to a new story discussing a previously unseen content. The number of buckets is finite, so whenever all buckets are occupied and a new story tweet streams in, the bucket that has the lowest timestamp of its most recent content update is freed up for reuse. Also the size of a bucket is limited. Whenever a new tweet is determined to belong to a full bucket, we simply remove the older half of the bucket to free up space. The assumption is that the tweets that the removed IDs refer to are not of importance (too far apart to be able to form a burst) since otherwise they would have been already reported as a burst.

An important Trident implementation detail lies in the necessity to first simulate the filling of the buckets. This is because Trident works on batches and, therefore, the state of the buckets gets updated only at the end of a batch. Let us consider the case when a new batch contains a new event as its first tweet, and the tweets following it are related only to that first tweet. Since that tweet is not placed in the bucket before the next tweet is processed, the next tweet would also be (incorrectly) identified as a new event and a new bucket would be assigned to it. This is a clear disadvantage of using batches in our algorithm, however, it pays off later when communication with a database system is required.

Finally, once in a while a fastest growing bucket will be detected. Before passing the burst content down the processing pipeline, the Burst Event Detector determines whether this bulk of tweets could be considered a burst by checking the time span within which the tweets were posted. The burst content is then consumed by the Disaster Detector component.

3) Disaster Detector: The task of the Disaster Detector is to find whether the captured event is an emergency situation and, if it is, where it is located. First, the keywords must be extracted and based on them, utilizing our disaster type dictionary, we identify the topic. Finally, if the topic matches a disaster situation, we localize it. At this stage, database query is necessary to access the tweet.

a) Extracting keywords: As keywords, we wish to find five words that best summarize the content of the burst. Our approach to finding them involves calculating TF within the burst. In other words, we only weigh each term positively for the number of times it occurs within the burst. The disaster dictionary is meant to contain words representative of a specific disasters. For our testing purposes, we populated our dictionary by representative keywords of a hurricane and snowstorm disasters. The datasets were compiled from querying Twitter's historical stream for the specific disaster types during their known occurrences.

b) Identifying disaster type: The topic is identified by matching any of the keywords to the part of the dictionary describing a given (predefined) topic. As an additional check, needed to prevent overlapping terms causing us to select the wrong topic, we weigh the keywords negatively relative to the number of times it occurs in the dictionary entry for other topics. The entries for topic, keywords, IDs of the contributing tweets and their text, location (if given) and users' information are saved to Cassandra. The database can be queried by other topologies (possibly spawned later to do retrospective analysis on the tweets) without affecting the performance of our system.

c) Geo-localizing the event: In order to inform only those crisis response agencies that are directly concerned with the occurring crisis situation, we need to be able to determine where the event is happening. Spacial grounding of the tweets in the burst requires to consider not only the coordinates, if given, but also the user's location as well as the places mentioned directly in the text of the tweet. This non-trivial task is carried out by the Geo-Tagger subcomponent we describe in our previous work [23].

4) Alert Component: Once a disaster event is detected and its location, type and representative keywords are determined, we pass the information to the Alert component. The keywords are used to start up new data collection sessions from Twitter both from recent history as well as from a real-time stream. Finally, the responsible operators can be notified who, for the duration of the disaster, are assumed to administer searches also from other social media channels using our CrowdSA application presented in [23]–[25].

IV. Results and Evaluation

Due to the difficulty of determining the "ground truth" and the appropriate metrics, a fully functional evaluation of our system (the correct and near real-time detection of a crisis situation) is beyond the scope of the present work. However, we devised three types of tests to show that our proposal of implementing the detection of crisis situations from Twitter content in Trident is reasonable. All tests carried out are run in an unsupervised mode, where we do not assume to know a priori the type of event being detected. The data collected for this purpose are from hurricane Iselle reaching Hawaii in August 2014, snowstorm affecting the US East cost in January 2016, and hurricane Patricia hitting Mexico in autumn 2015.

In the first test case, we published our collected historical data to the Kafka queue and measured the processing speed in terms of the number of tweets processed in a second. We were able to reach an average (for the three different datasets) processing speed of about 4000 messages per second. This is below what the full Twitter firehose could serve (on average, there are about 6000 messages posted on Twitter in one second[8]). However, after increasing the parallelization and utilizing a larger cluster, the performance can be boosted to surpass the firehose requirements.

In the second testing scenario, we evaluated the effectiveness of the burst detection capability and the keyword extraction by consuming the historical datasets from the hurricane Iselle. Our system could detect a number of bursts within the datasets occurring relative to the time of the day. Most importantly, the largest burst was detected on the 10th of August, which corresponds to our graphical inspection of the dataset.

For the third test, we fed the system by real-time Twitter streaming data. These account for about 1% of the Twitter Firehose. Since it was not expected to encounter any hurricane or snowstorm disaster during the run of the test, we let the system report on all bursts which it found, essentially skipping the Disaster Detector component. We observed that even with only six nodes, the system was performant in that it processed tweets fast enough without getting congested and falling behind.

All of the above tests were run on a six node virtual cluster with 8 cores each, 64 bit CentOS, and 16GB of RAM. The cluster was provisioned using Ambari[9] and runs Storm, Kafka broker, and a Cassandra server. The 1st node runs Nimbus (Storms master daemon) while the rest are the worker nodes (running Storm's Supervisors). Greater parallelization within Storm was selectively applied for intensive tasks such as the dot product calculation in similarity estimations. As a future evaluation strategy, we plan to experiment with different parallelization setups within Storm while processing the same data stream of tweets to better understand the capabilities of the distributed framework.

V. Conclusion and Future Work

In this paper, we reported on our work that focused on the problem of near real-time (online) Burst Topic Detection from streaming Twitter data in application to detecting newly occurring crisis situations. Our approach involves clustering tweet messages based on content similarity and implementing the algorithms in a parallel fashion using the Apache Storm

[8]http://www.internetlivestats.com/one-second/
[9]https://ambari.apache.org/

Trident API. Our results have shown that the use of First Story Detection in combination with capturing temporal bursts of similar messages streamed from a Twitter firehose, and mapping the captured events to a predefined disaster dictionary enables fast reporting of a crisis situation when implemented in a distributed fashion.

As a future work, we are interested in looking at the potential improvements if the structure of the social network is also considered in the similarity measurements. Additionally, we plan to compare the results of the unsupervised event detection (as currently performed by our setup) with a supervised approach in which we would use our dictionary to filter tweets before they are passed to the clustering component. Last but not least, we would like to experiment with incorporating a more sophisticated model for filtering out the tweets before bucketing. Namely, since in the case of a large burst, it might be more effective to also include the very near tweets in the processing, we'd like to use a dynamic threshold value updated based on the current state of the stream.

Acknowledgment

The authors would like to thank Matthias Steinbauer and the Telecooperation Institute of JKU Linz for providing cloud computing resources and support during provisioning our cluster.

This work has been funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under grant FFG BRIDGE 838526 and under grant AD WTZ AR10/2015.

References

[1] J. Kleinberg, "Bursty and hierarchical structure in streams," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, (New York, NY, USA), pp. 91–101, ACM, 2002.

[2] Q. He, K. Chang, and E.-P. Lim, "Analyzing feature trajectories for event detection," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, (New York, NY, USA), pp. 207–214, ACM, 2007.

[3] J. Weng and B.-S. Lee, "Event detection in twitter.," *ICWSM*, vol. 11, pp. 401–408, 2011.

[4] X. Gao, E. Ferrara, and J. Qiu, "Parallel clustering of high-dimensional social media data streams," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pp. 323–332, May 2015.

[5] G. Wu, O. Boydell, and P. Cunningham, "High-throughput, web-scale data stream clustering," in *Proceedings of the 4th Web Search Click Data workshop (WSCD 2014)*, 2014.

[6] K. XIANGSHENG, "Microblog mining based on cloud computing technologies: Mesos and hadoop.," *Journal of Theoretical & Applied Information Technology*, vol. 48, no. 3, 2013.

[7] J. Yin, A. Lampert, M. Cameron, B. Robinson, and R. Power, "Using social media to enhance emergency situation awareness," *Intelligent Systems, IEEE*, vol. 27, pp. 52–59, Nov 2012.

[8] J. Rogstadius, M. Vukovic, C. A. Teixeira, V. Kostakos, E. Karapanos, and J. A. Laredo, "Crisistracker: Crowdsourced social media curation for disaster awareness," *IBM J. Res. Dev.*, vol. 57, pp. 1:4–1:4, Sept. 2013.

[9] M. Nagarajan, K. Gomadam, A. P. Sheth, A. Ranabahu, R. Mutharaju, and A. Jadhav, *Web Information Systems Engineering - WISE 2009: 10th International Confee rence, Poznań, Poland, October 5-7, 2009. Proceedings*, ch. Spatio-Temporal-Thematic Analysis of Citizen Sensor Data: Challenges ann d Experiences, pp. 539–553. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[10] C. C. Aggarwal and K. Subbian, "Event detection in social streams," in *In Proceeding of the Twelfth SIAM International Conference on Data Mining*, (Anaheim, California, USA), pp. 624–635, SIAM / Omnipress, April 26-28 2012.

[11] H. Smid, P. Mast, M. Tromp, A. Winterboer, and V. Evers, "Canary in a coal mine: Monitoring air quality and detecting environmental incidents by harvesting twitter," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, (New York, NY, USA), pp. 1855–1860, ACM, 2011.

[12] T. Sakaki, F. Toriumi, and Y. Matsuo, "Tweet trend analysis in an emergency situation," in *Proceedings of the Special Workshop on Internet and Disasters*, SWID '11, (New York, NY, USA), pp. 3:1–3:8, ACM, 2011.

[13] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes twitter users: Real-time event detection by social sensors," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), pp. 851–860, ACM, 2010.

[14] F. Atefeh and W. Khreich, "A survey of techniques for event detection in twitter," *Comput. Intell.*, vol. 31, pp. 132–164, Feb. 2015.

[15] H. Becker, M. Naaman, and L. Gravano, "Learning similarity metrics for event identification in social media," in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pp. 291–300, ACM, 2010.

[16] S. Petrović, M. Osborne, and V. Lavrenko, "Streaming first story detection with application to twitter," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pp. 181–189, Association for Computational Linguistics, 2010.

[17] W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K. Wang, "Topicsketch: Real-time bursty topic detection from twitter," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pp. 837–846, Dec 2013.

[18] Q. Diao, J. Jiang, F. Zhu, and E.-P. Lim, "Finding bursty topics from microblogs," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, (Stroudsburg, PA, USA), pp. 536–544, Association for Computational Linguistics, 2012.

[19] S. Liu, D. Shaw, and C. Brewster, "Ontologies for crisis management: a review of state of the art in ontology design and usability," in *Proceedings of the Information Systems for Crisis Response and Management conference (ISCRAM 2013 12-15 May, 2013)*, 2013.

[20] D. De Wrachien, J. Garrido, S. Mambretti, and I. Requena, "Ontology for flood management: a proposal," *Flood Recovery, Innovation and Response III*, vol. 159, p. 3, 2012.

[21] Y. Matsuo and M. Ishizuka, "Keyword extraction from a single document using word co-occurrence statistical information," *International Journal on Artificial Intelligence Tools*, vol. 13, no. 01, pp. 157–169, 2004.

[22] W. X. Zhao, J. Jiang, J. He, Y. Song, P. Achananuparp, E.-P. Lim, and X. Li, "Topical keyphrase extraction from twitter," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 379–388, Association for Computational Linguistics, 2011.

[23] A. Salfinger, W. Retschitzegger, W. Schwinger, and B. Pröll, "Crowd sa – towards adaptive and situation-driven crowd-sensing for disaster situation awareness," in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Inter-Disciplinary Conference on*, pp. 14–20, IEEE, 2015.

[24] A. Salfinger, S. Girtelschmid, B. Pröll, W. Retschitzegger, and W. Schwinger, "Crowd-sensing meets situation awareness: A research roadmap for crisis management," in *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pp. 153–162, IEEE, 2015.

[25] A. Salfinger, W. Retschitzegger, W. Schwinger, and B. Pröll, "Mining the disaster hotspots – situation-adaptive crowd knowledge extraction for crisis management," in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2016 IEEE International Inter-Disciplinary Conference on, in press*, 2016.