Distributed Real Time First Story Detection using Twitter
Informatics Research Proposal

Pengfei Gao, s1144374
Supervisor: Miles Osborne

April 3, 2012

Abstract

In the recent days, the social networking media has been widely used by more and more people. There is a growing need to develop a powerful and efficient system tools to mining the useful and important information from the huge amount of data generated by users from those social networking media. The aim of this project is to develop such a real-time First Story Detection system to detect current hot events from the real-time stream of Twitter. The project will use Apache Distributed Stream Computing platform(S4) and locality-sensitive hashing method to process continuous tweets stream and detect the hot events from the noisy stream.

1 Purpose

Twitter is a popular real-time information network that users can present and find the latest ideas, stories, news and opinions on what they interested. For example, when the news the pop star Michael Jackson's death came out, 22.64% of the tweets posted contains the phrase "Michael Jackson", which enables the detection of the current hot events promptly.

New event detection includes topic detection and tracking. A story is a topically cohesive segment of news that includes two or more declarative independent clauses about a single event. Given a sequence of stories, the goal of FSD is to identify the first story to discuss a particular event. In this context, an event is taken to be something that happens at some specific time and place, e.g., an earthquake striking the town of Szechuan in China, 2008.

2 Background

First Story Detection deals with spotting breaking news [1]. For example, as soon as an earthquake happens, we want to know about it. We do not care about follow-up stories. Now, Twitter can be an excellent source of first stories and we have shown how they can

be spotted using scalable algorithms based upon locality sensitive hashing.

The traditional approach to detect the first story is using the vector space model which represents all the tweets as linearly-independent vectors and then projects them in some high-dimensional space [6]. The coordinates of the space indicates the frequency of the particular words appeared in the tweets. Each new tweet is then compared to those existed ones and rank the similarities of those tweet vectors using dot product or cosine of the angle between vectors. If the new tweet's similarity to the closet tweet is under a certain threshold, the new tweet is claimed to be a first story. This model is very popular in the topic detection and tracking area.

Locality sensitive hashing (LSH) is a method of hashing input data and put the collision data into the same bucket. As the buckets size is much smaller than the total input data, the similar items will have high probability being put to the same bucket. When new tweet is hashed into a bucket, all the tweets in the same bucket will be compared and the most similar tweet will build a connection to the new one. [5] reports that simply using LSH for the closest neighbour search for the FSD performs poor and results in a high variance which a query point is hard to find the true nearest neighbour when it lies far away from all its neighbours. They introduce a modified LSH method which can reduce the variance and improve the accuracy. When the LSH approach claims a document is different from others, it will search through the inverted index of those existed documents with a limited number of the most recent document. This method works well and we will use it for this project.

In the context of first story detection, we need to detect the current hot topic from the tweets that multiple users just update. The users may update their tweets at the same time. Therefore our FSD system should be distributed and scalable in order to track and analyse those users tweets at the same time. Although each tweet is just 140 character long, a user can tweet 100 or even more times a day. Especially some emergent important events happened such as Tunisian revolution, million people will tweet or retweet the corresponding news in the specific period of time. That leaves us a lot of tweets to track. However, we can not store all the tweets in the memory nor compare the new tweets to all the existed tweets. Thus, for an unbounded tweets streaming, the streaming FSD system should track each tweet and then make decisions whether the topic is seen before or a new one and then process the next tweet. The decision should be made in the limited time and space for each tweet. Such a system with all above features can be used to detect the first stories in the real time from an unbounded streaming of tweets from the Internet.

As most of the first stories are just currently hot in a while and then being forgotten gradually, we need to limit the number of tweets in the bucket to a constant. We use a First-In-First-Out method here to keep the size of the bucket. In addition, we also limit the number of tweets to be compared when a new tweet comes. [5] compares each new tweet with less than 3L tweets which it collided with.

People use Twitter to find the latest stories, ideas and news about what they interest in and also use it to express their opinions and share their daily life with their friends. In

this sense, most of the tweets are not real stories but personal things, advertises and even spam. Therefore, simply using the FSD system to track and analyse every tweet will lead to an incredible number of new stories and most of this are not real first stories which we do not want. Our goal is to find the current significant first stories in the real time. To achieve this goal, [5] suggest a process that reduce the number of the non-important events. First they detect the a twitter's nearest neighbour and assign a score to is using the cosine distance. [3] introduces a links relation to analyse threads of tweets: Tweet a links to tweet b if b is the nearest neighbour of a and

$$d = 1 - cos(a, b)$$

where t is a threshold defined by user. Then, for each tweet a, if $d < t$, they will assign a to an existing thread containing the its nearest neighbour. Otherwise, they will create a new thread and a will be first tweet for thread. As changing t will affect on the granularity of threads, [5] find t belong to $[0.5, 0.5]$ results are much better.

Now we just need to compare the growing speed of those threads as a highest growing speed indicates the news of a new event is spreading widely and quickly. Thus, for each time interval we only capture the fastest growing thread and this thread represent the currently hottest first story.

S4 , inspired by MapReduce, is a distributed processing system which spreads computation over a network of machines. It is a general-purpose, distributed, scalable, pluggable platform that used for processing continuous unbounded streams of data. Tasks communicate with each other via remote procedure calls using key-value pairs. Because of this, processing is scalable and fast.

[4] define a stream as a sequence of events in the form (K, A) where K is keys and A is the attributes. S4 is designed to consume process such a stream. In S4, Events are arbitrary Java Objects that can be passed between PEs. Adapters convert the external incoming data streams into Events and then dispatched in named streams which identified by a key stream name. Then keyed data events are routed with affinity to Processing Elements (PEs), which consume the events and emit one or more events which may be consumed by other PEs, or publish results, possibly to an external data store or consumer.

Processing Nodes (PNs) are the logical host to PEs which are responsible for listening to events, executing operations on the incoming events, dispatching events with the help of the communication layer, and emitting output events. Key-less PEs usually serve as entry point PEs: That is, the PEs that receive events from the outside world. This is because an adapter typically does not choose an S4 node based on a key in the event, but randomly (or in some other fashion that evenly distributes the events amongst the S4 nodes).

3 Methods

We will define a PE type called $TweetReceiverPE$ which is configured to listen to the tweets stream. Each incoming tweet is event (I define Tweet object here) that has no key. So $TweetReceiverPE$ is a keyless PE object that receives the new tweets and do some text preprocessing on Tweet object, such as tokenization, stemming, removing stop-words and some Twitter default characters (i.e. @).

After the preprocessing, each tweet is left with several important words. $TweetReceiverPE$ will create a vector of these words representing this tweet and emit a new event of type $TweetEvent$, keyed on the hashes of these words vectors using LSH. $TweetBucketPE$ objects listen for the $TweetEvent$events emitted with key hash and put them into the corresponding buckets if their hashes are the same as the existed events in the bucket. For example, the $TweetBucketPE$ object for key hash receives all events of type $TweetEvent$ keyed on hash.

When a $TweetEvent$ event for key comes, S4 looks up the $TweetBucketPE$ object using the key hash. If the $TweetBucketPE$ object is existed, the PE object is invoked and the $TweetEvent$ will be saved into bucket. Otherwise a new $TweetBucketPE$ object is created. Whenever a $TweetBucketPE$ object saves an $TweetEvent$, it sends to updated information to the $TweetSimilarityPE$ object. The key of the $TweetSimilarityPE$ object is a random integer. Once $TweetBucketPE$ object chooses a $similarityID$, it uses the $similarityID$ in its whole life cycle.

The $TweetBucketPE$ will calculate all the similarities for the new incoming $TweetEvent$ object with other existed $TweetEvent$ objects and find its nearest neighbour. When $TweetBucketPE$ object has found the nearest neighbour of the new $TweetEvent$ object, it sends a message including the $TweetEvent$ object and its neighbour to the $TweetThreadPE$ object. $TweetThreadPE$ object checks whether the new $TweetEvent$ object links to its nearest neighbour based on the threading function mentioned in [5]:

$$d = 1 - cos(a, b)$$

If the distance d is smaller than the threshold t, the new Tweet will be assigned to an existing thread which its nearest neighbour belongs to. Otherwise, $TweetThreadPE$ will create a new Thread object and the new Tweet will be its first tweet. The threshold t is set equal to 0.5 according to [5].

After the threads being updated, S4 will return the growing speed of each thread (simply the number of $TweetEvent$ object each thread contains) and we are interested in those threads growing fastest. Therefore, in each time interval we just need to analysis the top fastest growing threads and these will be considered the first stories.

4 Evaluation

The project is to implement a distributed system to detect the first stories from the provided tweets using Java and S4 framework. The feasibility of this distributed system that

can be deployed to a cluster of servers and detect the some hot topics from a provided tweets stream is the basic evaluation metric.

[5] indicates that those tweets are already labelled by two human experts as either Event, Neutral, or Spam. For those tweets labelled as an Event, it is clear for us that what happened exactly without any prior knowledge and the related tweets are significant important. [2] stated that average precision is a usual evaluation metric in normal tasks that the returns are judged by the relevance. As we have the labelled tweets, this project will be evaluated by computing the average precision on the tweets to be correctly detected based on their labels. For example, in one specific thread, event tweets are detected to be relevant, neutral and spam tweets are treated as non-relevant.

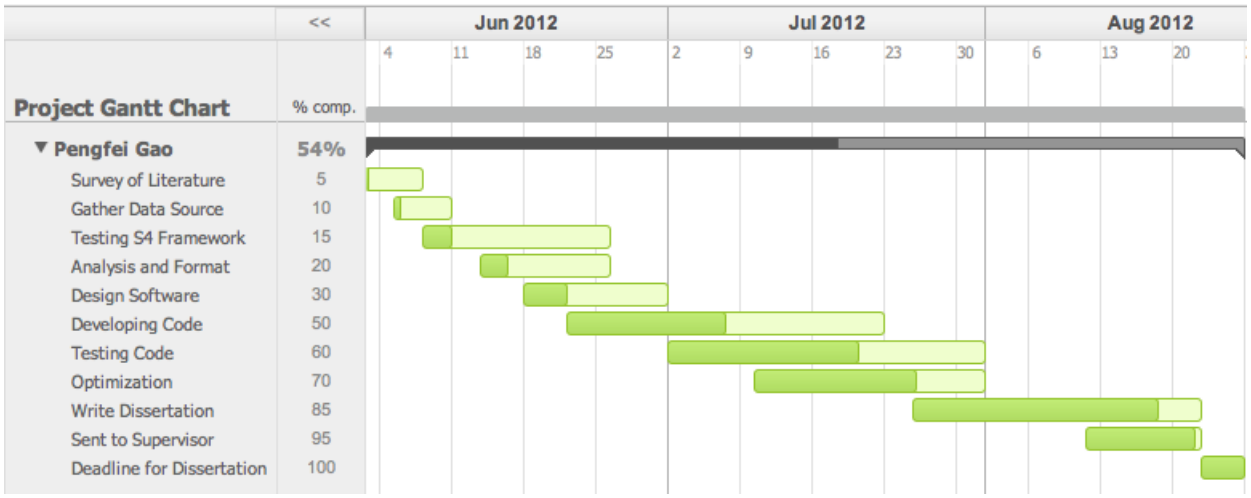
The performance of this first story detection system is another import issue for the evaluation metric. We will try to evaluate several different ranking methods for the outputs (threads) in order to detect important events from a noisy Twitter stream. The rate of the stream will be controlled differently to test the robustness and scalability of the system. The server cluster will be configured by ZooKeeper.

5 Outputs

The output of this project will be a distributed real time program which detects the breaking news from Twitter. The program is implemented based on S4 framework using Java. It will run at several computers in DICE systems. The program will read some existed tweets and find the hot topics in those tweets. The topics will be some events described by some words.

6 Workplan

The final section identifies the workflow of my project along with expected completion dates.



References

[1] James Allan, editor. *Topic detection and tracking: event-based information organization*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[2] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison Wesley, 1 edition, February 2009.

[3] Ramesh Nallapati, Ao Feng, Fuchun Peng, and James Allan. Event threading within news topics. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 446–453, New York, NY, USA, 2004. ACM.

[4] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170 –177, dec. 2010.

[5] Saša Petrović, Miles Osborne, and Victor 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, pages 181–189, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[6] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and online event detection. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 28–36, New York, NY, USA, 1998. ACM.