Group 4: Real Time Social Media Data Integration and Analysis Application

Chen Li (cli24@uci.edu)
Shuying Lai (shuyinl2@uci.edu)
Bálint Tillman (tillmanb@uci.edu)

Introduction

In recent years there has been an increased interest in processing and responding to events in real time by exploiting online social media generated content. A prominent source is Twitter, where users can use up to 140 characters in short messages (tweets) along with pictures; GIFs and even 30 second long videos.

In this project, we focus on building a pipeline to deliver real time information according to given geographic constraints. Our design uses modern technologies that are capable to handle both streams of information and historical data at the same time. However, it is important to understand that there will be always some network latency which is unavoidable. For example: Twitter streams arrive from Twitter itself and depending on the location of our service, we will experience latency; similarly from our service to the end users there will another layer of latency. We can cheat the human perception of real time, as long as these latencies are relatively small.

Our work is in strong contrast with other projects where complex models are trained using machine learning techniques. In these projects the original data could come from the Twitter Streaming API or crawled in some manner over a period of time, but training happens mostly offline or results come with a significant delay. We will show a related work with comparable delay to our basic approach using Storm.

This report gives an overview of our considerations to build a prototype system and technologies to use. Once these components and frameworks are chosen, it is possible to design and implement the final pipeline. In our proof of concept system, we focus on delivering real time sentiment analysis (depending on latencies in communication) through a web interface to users.

Related work

In recent years there have been many papers considering the twitter streaming API and to utilize data available from it. However many of these work are not real time processing, a good example for this issue can be found in [2]. Here the authors build a model to describe users, their location and the diversity in recognizable linguistic features. They claim to be scalable, but their method is potentially far from real time processing, since they only consider the data from the streaming API over a period of time as a single dataset.

The only comparable work that we could find is from McCreadie et al. [1]. In this paper they consider to detect events using clustering methods with low latency. This is computationally more expensive than our approach but it is built on top of Storm (see in next section) which in principle only handles streams of data and it is not designed to process

other datasets (historical data processed in batches). Even in this paper we only see synthetic tests but at least the authors provide measures that the approach is scalable.

Background

In this section we provide general information about some of the systems, frameworks we have considered to build our prototype which are designed for real time applications.

Spark

Spark revolves around the concept of a resilient distributed dataset (RDD) [3], which is a fault-tolerant collection of elements that can be operated on in parallel. RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset.

All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. Spark is faster than hadoop because it does not have to write and read the intermediate results of transformations.

However Spark in itself is not designed to work well with streaming data. In [4] an extension was introduced to the core Spark API which become Spark Streaming. This extension enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming provides a high-level abstraction called discretized stream or D-Stream, which represents a continuous stream of data. D-Stream treats a streaming computation as a series of deterministic batch computations on small time intervals. They store the computation results in resilient distributed datasets (RDDs). D-Stream groups together a series of RDDs and lets the user manipulate them through various operations. Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. In our project, we ingest data from Twitter streaming API, and process it using mainly map transformation.

Storm

Storm is a real time fault-tolerant and distributed stream data processing system developed at Twitter [5] and later became part of Apache open source project library [6]. It is designed to be scalable, resilient, extensible, efficient and easy to administer. The basic data model for storm consists of streams of tuples flowing through topologies. The topologies is a directed graph in which vertices represent computation components while edges represent data flow between these components. Vertices further divided into spout and bolt. Spout can be viewed as data sources for tuples while bolts can be viewed as processing units process data come from upstream and output it to the downstream.

Using this data model, storm runs on distributed environment. Similar to Hadoop, it has a master node called Nimbus, which is responsible for assign and manage the execution of the topology, if Nimbus is down, new topologies can not be submitted. And also worker nodes which run one or more worker processes. Further divided, each worker process runs one or more tasks, in which the actual work for the topology is done and consisted of two types of threads: worker receive thread and worked send thread. From this structure we can see that Storm provides several levels of parallelism. When data is shuffled from one vertices to another in the topology, there are several types of petitioning strategies that can be used for the programmer, including shuffle grouping, fields grouping, all grouping so on and so forth. Also, each worker node runs a supervisor to communicate with Nimbus and monitor its own health using different types of events. All coordination between Nimbus and Supervisors are through Zookeeper. Yet another important property in Storm is its processing semantic. It has two kinds of semantics: “at least once" and "at most once”, which guarantee the time each tuple will be processed in the topology.

Storm has many applications in Twitter support real time data-driven decisions and is also developed and used in many other organizations and scenarios. It is one of the most popular real time data processing framework, has a good integration with Hadoop ecosystem, and with an active community. However it is not directly designed to process batch data (in our case historical or external datasets), only data that arrives as a stream.

Play Framework and Akka

Play Framework [10] is an open source high velocity web framework for Java and Scala, using the popular model-view-controller architecture. The major difference between Play and other web framework is that Play is fully RESTful, i.e., stateless without traditional Java EE session , and supports fully asynchronous model built based on Akka, which makes it scales simply and predictably, and easily supports truly real-time web application.

Akka is a toolkit and runtime for building highly concurrent, distributed and resilient message-driven application on the JVM [12]. Built on Actor Model, Akka raise the abstraction level and provide a better platform to develop scalable, responsive and fault-tolerant applications. It provides a concept called Actor, which gives you a high-level abstractions for distributed applications, and it also adopts the “let-it-crash” semantics to support fault-tolerance. Akka provides scalable real-time transaction processing, which can be used in applications like Online Gaming, Banking System and Social Media.

AngularJS

As our design will use a web interface as the front end to represent our computations, we have looked into several javascript libraries (such as jQuery, ExtJS) to develop an easy to use and maintainable interface. We have chosen AngularJS, because it supports both WebSockets and two-way data binding [7,9]. This results in less lines of code and makes our application “real time” out of the box. Similar reasons lead others to make the same choice as we did to build real time web applications, one such application can be found in [8].

AngularJS uses a version of Model-View-Controller design, but the model and view can be tightly integrated using two-way data binding. The main functionality is that whenever a change happens in the model/data, the view/UI automatically updates according to a template. There is no need for other functions to be implemented, this is handled by AngularJS.

System overview

Our system consists of four parts: the real-time twitter streaming, spark cluster, Play/Akka web framework, and front-end AngularJS, as shown below:

We first fetch real-time tweets from Twitter Streaming API, then combine with the historical data, we perform sentiment analysis on a two-node spark cluster built on Amazon Web Services. After getting the sentiment analysis results, we are using Play framework's Actor System to talk to front-end AngularJS using websocket. Finally, using AnguarJS's two-way data binding, the results are showed on the openstreetmap in the form of heatmap. The whole data pipeline is non-blocking, scalable and fault-tolerant.

Implementation details

Twitter Streaming API

Twitter provides two kind of API for fetching tweets: Streaming API and restful API. On the one hand, the streaming api provides a truly real-time twitter stream, tweets come as soon as they are generated by the user, however, it doesn't support advanced functionalities such as fetching tweets in a specific area and so on. On the other hand, the restful api is not "truly real-time", we send request every 5 or 10 seconds to fetch a certain amount of tweets, it supports fetch by location and keywords. Since in our application we want to only fetch tweets in a specific location, and several seconds delay is tolerable, we choose the restful API. We have made this choice due to the small amount of tweets with geotags in the Streaming API.

Sentiment Analysis

Here we provide detailed information about how we have approach to compute sentiment scores for every tweet using Spark. Tweets can arrive both from through streams and loaded from historical data. Computing sentiment score has the following steps:

1. Filter RDD of twitter4j.Status with desired location. (Twitter4j.Status, provided by twitter4j package, A data interface representing one single status of a user.)
2. Map RDD of twitter4j.Status to RDD of JsonObject with all the fields we need, including tweet text (trimmed and special character removed), timestamp, location, and tweet id. (We choose to represent a tweet as a JsonObject because a JsonObject can be serialized and is convenient to parse on frontend.)
3. Map RDD of JsonObject to RDD of JsonObject with sentiment score added.
4. Call Spark function collect() to collect data from distributed clusters to master, and return data to frontend. (collect() function should be used with caution because it gathers all data to a single node and might encounter storage shortage. We use this function because each time we only have a stream of tweets in 10 seconds.)

Web Application

We use Play/Akka framework as the backend and AngularJS as the front-end, they are connected using websocket. In the backend, an application controller handles the basic logic of the application, such as fetching tweets and performing sentiment analysis. For each user login to our system, we build a user actor (based on akka system model), which handles user's interaction such as searching for their interested places and then send requests to the application's controller. After getting the results, the user actor pass it to the front-end using websocket. Since all the data transformation is using Akka's message passing and websocket, the whole data pipeline is non-blocking.

In the front-end, AngularJS is listening on the websocket, when the new result arrive, Angular's two-way data binding will automatically trigger the update, which using angular-leaflet directive (a specific directive designed for the AngularJS and leaflet map) to draw makers and heatmap on the openstreetmap.

Results

In the following table results are shown on the processing time for tweets in our historical data which consists of 3,711,943 tweets in total and 377,183 tweets with geotags. The filtering is only relevant for historical data, but the sentiment analysis results apply to incoming streams as well.

Location	# twitter	Get location data time	Get sentiment data time
Irvine	771	18.917491 s	7.708516 s
Los Angeles	14733	20.777380 s	143.480886 s
NYC	21012	21.155872 s	200.305145 s

For the three above locations first we show the number of tweets available in our dataset, then the time required to apply filters in Spark to focus on a given part of the dataset. Here we can only see a slight variation in time given the final number of tweets for each location. Filtering heavily depends on disk accesses which contributes to most of the overall time required.

In the final column, we present our sentiment analysis running time that is approximately linear in the number of tweets processed. This observation is consistent with our knowledge about spark.

In our system it takes <10ms to compute a sentiment score per tweet. In comparison with [1], we consider this a good result, since their analysis has multiple steps and depending on the approach, it is up to 10x slower. We do understand that their presented analysis is more complicated, but it is a good measure for scalability.

Future work

We would like to see how our approach and [1] are performing in the wild. This is difficult in reality due to the low number of tweets using geotagging. In principle our approach is fully scalable to handle the full Twitter Stream and performs closely to related work.

In the future we would like to see more complicated analysis similar to [1], but possibly using online machine learning techniques or other online algorithms. In these scenarios a given model is gradually trained over incoming tweets and prediction can be made with increasing accuracy as the model trains. In these algorithms an update step can be efficiently implemented, thus we would maintain the real time aspect of our system.

In addition it would be interesting to see similar analysis for Facebook updates and other similar real time social media updates and eventually a combination of these.

Conclusion

In this project we have considered different technologies to build an end-to-end real time social media processing pipeline showing interesting sentiment analysis to users. First we have shown our understanding in current distributed computing systems that can easily process incoming stream of information such as Spark and Storm. In the next step we have picked a scalable framework (Play Framework) to deliver messages to web users using websockets in close to real time.

Our implemented system was deployed to AWS and we have compared our batch processing results to the closest related work which shows that our sentiment analysis can be shown in real time. This proof of concept opens the possibility to explore more complex analysis while keeping the balance between complexity and real time responses.

References

1. McCreadie, Richard, et al. "Scalable distributed event detection for twitter." *Big Data, 2013 IEEE International Conference on*. IEEE, 2013.
2. Hong, Liangjie, et al. "Discovering geographical topics in the twitter stream." *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012.

3. Zaharia, Matei, et al. "Spark: Cluster Computing with Working Sets." *HotCloud* 10 (2010): 10-10.
4. Zaharia, Matei, et al. "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters." *Presented as part of the*. 2012.
5. Toshniwal, Ankit, et al. "Storm@ twitter." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014.
6. Goetz, Taylor, Blog post: http://hortonworks.com/blog/brief-history-apache-storm/
7. Green, Brad, and Shyam Seshadri. *AngularJS*. " O'Reilly Media, Inc.", 2013.
8. Chaniotis, Ioannis K., Kyriakos-Ioannis D. Kyriakou, and Nikolaos D. Tselikas. "Proximity: A Real-Time, Location Aware Social Web Application Built with Node. js and AngularJS." *Mobile Web Information Systems*. Springer Berlin Heidelberg, 2013. 292-295.
9. AngularJS reference: https://docs.angularjs.org/guide/databinding
10. Play Framework: https://www.playframework.com
11. Wikipedia about Play Framework: https://en.wikipedia.org/wiki/Play_Framework
12. Akka Framework: http://akka.io