Data Centric Systems and Networking

Using Apache Storm to monitor location-based sentiments

Michael Schaarschmidt (mks40)

1 Introduction

This project aims to build a simple location-based sentiment tracker of live twitter feeds using Apache Storm. Twitter has become a main target for analysing social trends and sentiments [1, 20] in real-time. Since its launch, Twitter has even emerged as an independent source of breaking news (e.g. during the Arab spring [19]) and major news outlets [2] use twitter for live journalism. Twitter also provides access to its tweets through a streaming API which constitutes a comfortable starting point for analysing incoming streams of tweets. Applying a filter to a stream enables us to track evolving stories about certain topics or tweets from locations. However, the limitations of Twitter's streaming API make it necessary to perform some parts of the filtering in our application, rendering it a suitable use-case for Apache Storm. In section 2, we will provide a brief introduction to Storm and discuss related work. Sections 3 and 4 introduce the concept and implementation. Section 5 and 6 conclude the report with an experimental evaluation and an outlook towards improvements.

2 Overview of stream processing

2.1 Apache Storm

Apache Storm is an open source distributed processing engine that is targeted towards computations on continuous data streams [24]. Storm is used by a wide range of companies and notably by Twitter [26] to provide scalable and fault-tolerant real-time services. To use Storm, developers first have to specify a topology. Topologies are similar to jobs in MapReduce [6] with the difference that a topology does not finish processing data until it is turned off. A Storm topology is a directed, acyclic graph that consists of "spouts" and "bolts". Spouts are sources of incoming data streams, e.g. twitter feeds, click-streams, server event logs etc. Bolts receive data from spouts in the form of tuples, process them in parallel and emit them as new streams to other bolts.

Much like a Hadoop [21] cluster, Storm is organized by a master node that coordinates worker nodes. The transactional semantics of the original Storm implementation guarantee that each incoming tuple is processed by the topology at least once or at most once, depending on the configuration. At most once semantics can be achieved by discarding a tuple if it fails to process for some reason (e.g. exception raised or worker node failure). On the other hand, at-least-once semantics require the system to keep track of emitted tuples and replay them when they are not acknowledged within a certain time window. Finally, exactly-once semantics can be achieved by using the Trident [16] abstraction on top of Storm, which uses unique transaction ids and ordered state updates to ensure replayed tuples are only processed once.

The master runs a "Nimbus" daemon that assigns jobs to individual machines and handles failure by restarting workers or reassigning tasks. Master fail-over is handled by an Apache Zookeeper [25].

2.2 Related work

Apache Samza [23] provides very similar functionality to Storm. Samza has a topological model similar to Strom consisting of stream consumers and messages. However, it does not provide as many options regarding transactional semantics as Storm: Currently, only at-least-once processing is supported, other modes are planned. An interesting feature of Samza is its approach to state management: Samza tasks come with an embedded-key value store that allows for high-throughput reads and writes even when processing data that exceeds individual memory. Storm can provide checkpoint semantics to remote databases with Trident, but is not suited to handling large amounts of state data on worker nodes. Specifically, processing is slowed down by the network traffic incurred by reading and writing to a remote database. Ultimately, both frameworks provide fairly similar functionality and choosing the more mature model seems to be sensible.

Naiad [18] is another system capable of processing streams as well as incremental and iterative tasks. Naiad introduces the concept of timely dataflow, which consists of stateful vertices that send and receive messages and notifications using timestamps. Providing an in-depth explanation of the complete Naiad stack is beyond the scope of this report so we will confine us to the fundamental differences between Storm and Naiad. Storm without Trident does not offer state management. As outlined above, bolts can hold objects representing a state in memory, but there is no inherent concept of stateful computation. Naiad focusses on reads and writes to mutable states on vertices, based on the observation that stateless vertices have to send their entire state to the next worker, which is particularly inefficient on incremental algorithms. The timely dataflow model is a much more powerful abstraction than Storm in the way it focusses on efficiently representing change and its ability to quickly coordinate stages of a dataflow cycle in a cluster. Storm does not even attempt to exploit any insights towards typical computations and provides its topology as some kind of agnostic computation pipeline, which makes Storm arguably easier to use.

Finally, Apache S4 [22] is yet another stream engine. However, there do not seem to be any updates to the project in the last two years and only little information.

3 Concept

The goal of this project is to demonstrate how Apache Storm can be employed for live capturing and analysis of twitter data. However, the twitter streaming API poses some restrictions to the design of the application [27]. First of all, a single account can only have one connection to the streaming API (i.e. only one spout). Streams can be filtered for topics, locations, languages and so forth. However, the streaming API does not allow the intersection of filters: If multiple filters are added to a single stream, the union of those filters is returned. This makes it necessary to do complex filtering on the side of the application. It should also be noted that the incoming stream does not comprise all live tweets, but rather a sampling. The so-called "firehose" mode guarantees delivery of all live tweets but requires paid access.

Considering these restrictions, we propose the following concept: First, the application is supplied with the bounding box of a larger region (e.g. the USA, UK or Europe). Topic filtering is not applied to the stream but done in Storm: this way, the application can track all (sampled) tweets in the area and collect relative frequencies of tweets that match certain topics. The main focus of the evaluation will concern the use of probabilistic data structures for space efficient processing. As discussed earlier, Storm is not designed towards stateful processing (Trident can handle stateful semantics at the cost of performance). In the scope of this project, we will investigate the tradeoffs related to using approximation techniques when analysing live data.

4 Implementation

4.1 Architecture

The application was implemented in Java , allowing for easy deployment and providing access to useful libraries such as a binding to the Twitter streaming API.

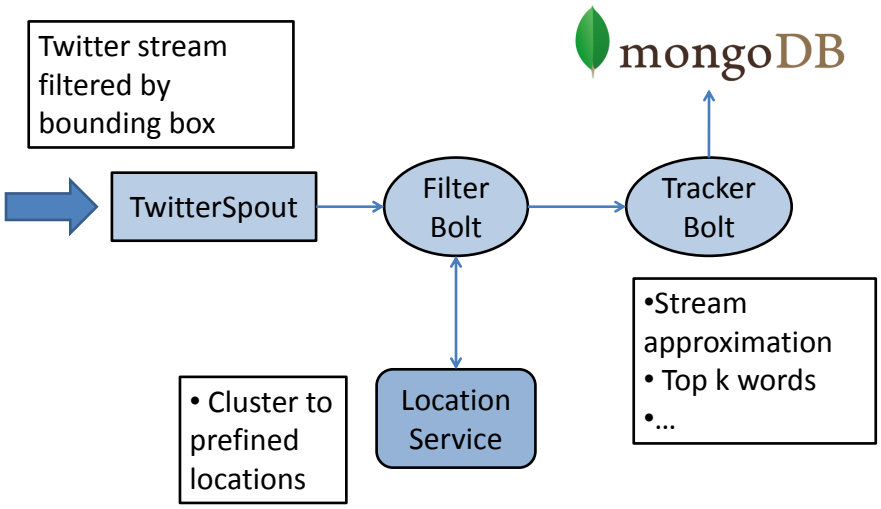

Figure 1: Topology

Figure 1 provides an overview of the major modules of the tool. Live stream data (or historical data) is fed into the *TwitterSpout*. A preliminary filtering on the stream limits inputs to certain geographical regions (e.g. USA). Next, tweets are clustered towards a predefined set of locations within the bounding box to analyse the geographical distribution of an emerging topic. Consider the recent example of public outrage over the decision of a grand jury in New York City over the death of a citizen (Eric Garner) during an arrest [15]. It would be interesting to see how a local event garners nation-wide attention in the social space. We can thus provide the location service with a few cities in different regions of the US (e.g. on the west coast, in the south, in New York City itself etc.). Incoming tweets will then be clustered according to these regions by the *FilterBolt*. Next, the *TrackerBolt* receives the clustered tweets and matches them towards a topic. Finally, we do live analysis as described in the next section.

4.2 Efficiently analysing streams

In this section, we are introducing the actual evaluation of tweets in the *TrackerBolt*. First, we use a Count-Min-Sketch [5] to keep track of the counts of all words that occur within tweets matching our topic filter. Count-Min-Sketches are parametrized through the width w and depth d of a two-dimensional array. Each row is associated with a hash-function so that incoming values are mapped to one position in each row, incrementing a counter on that position. The frequency of a value can then be estimated over the minimum of all related counters with a certain false-positive rate due to collisions. A nice property of the Sketch is that we can formally argue about the false-positive rate. The estimation error $\epsilon \leq 2n/w$ has a probability $\delta = 1 - (1/2)^d$. The Count-Min-Sketch only keeps track of frequencies, but not words. We thus implemented a heavy-hitters data structure that manages of a heap of the top k most frequent words based on counts estimated in the Count-Min-Sketch.

Finally, we need an efficient method of analysing words. For a simple sentiment analysis, this can be mostly reduced to the problem of comparing two sets of strings against each other.

When analysing tweets, we want to filter out stop words (e.g. "a", "to"). Finally, we can start approximating a sentiment by testing words in matching tweets for membership in a curated list of positive and negative words [4]. If these lists are small, we can simply load them into hashsets for membership queries. If space efficiency is of primary concern, we can also employ Bloom filters [3] for sub-linear space membership queries with certain error.

4.3 Fault tolerance

Since the tool relies on live streaming data and does not filter old tweets, it would be useful to be able to stop and restart without losing the collected statistics. This is especially an issue because experiments were only run on a personal computer and not on a stand-alone server that can stay online. The system thus contains a simple recovery tool: whenever a specified number of new tuples has been processed by the last bolt, the current results are serialized and stored to MongoDB [17]. MongoDB is an open source schema-free document database that has native support for geospatial indexing. Not having to define and change schemas and being able to store arbitrarily nested data-structures makes it an ideal candidate for an evolving application. On start-up, all previous statistics on a given topic are read into the Count-Min-Sketches.

5 Evaluation

5.1 Estimating the scale

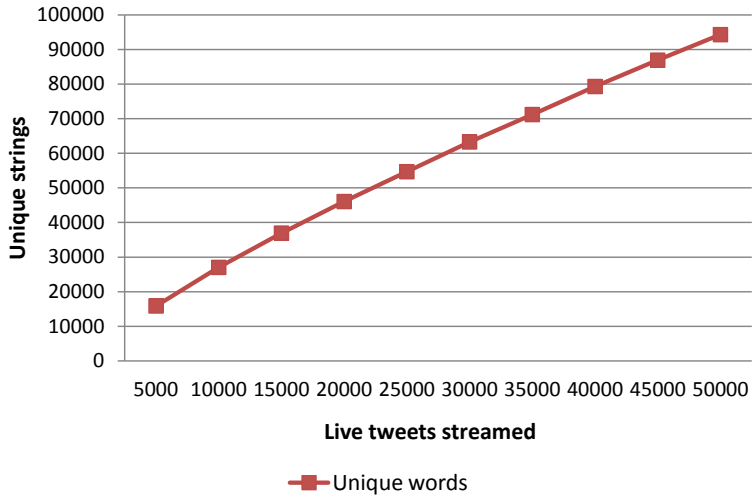

Figure 2: Counting unique strings in live tweets

A first question related to the use of probabilistic data structures relates to uniqueness of words. Figures 2 and 3 show how the number of unique strings seen increases over time on both live and historical data from a twitter corpus [10]. To this end, the tool includes replay functionality that allows us to stream historical tweets from a file with a defined data rate. Even after scanning a million historical tweets, we still observed a surprisingly linear increase.

Note that since we do not check for different forms of the same word (i.e. singular/plural etc.), the measured number is inflated. However, clustering all words would introduce much additional computational efforts for a single machine. Since we can safely assume that most words are infrequent [9], it does not make sense to keep counts for all the mostly infrequent words. Additionally, we cannot efficiently query very large data structures without making use of indexing techniques.

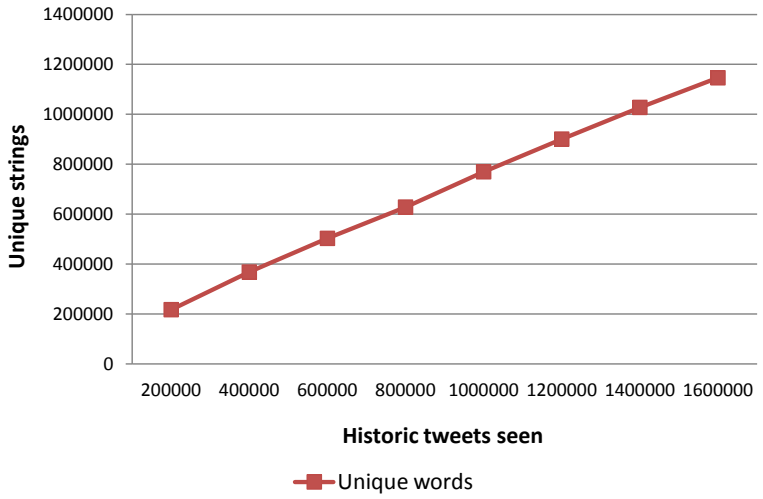

Figure 3: Counting unique strings in historic corpus.

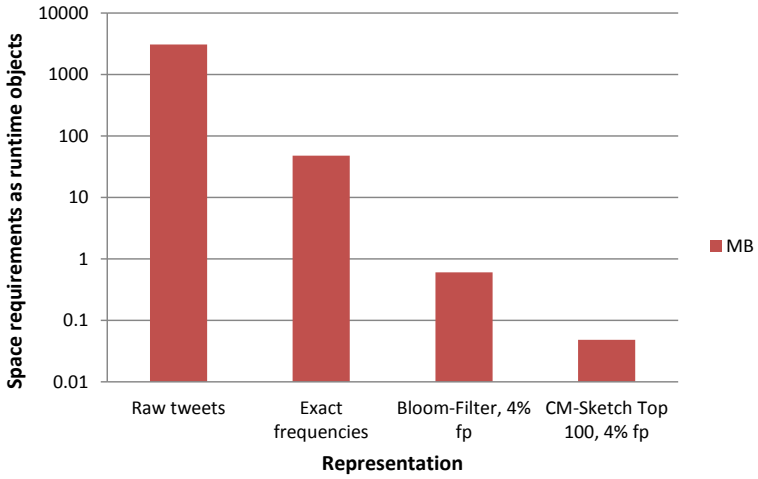

Figure 4: Runtime memory required for different representations. Calculations are based on an average word size of 5 characters in the English language and on a Zipfian distribution [5, 7].

Figure 4 illustrates memory requirements based on those observations, underlining the need for efficient approximation if we were to stream the firehose.

5.2 Performance

Next, we evaluate the performance of the approximation we employ. All experiments were carried out on a commodity laptop with 4GB RAM and a 2.5 GHz i5 core. Using the streaming API, CPU loads were constantly low (sub 10%). We attribute this towards being bottlenecked by the sampling rate of a single streaming connection, which rarely exceeded 40 tweets per second in our experiments (as discussed in the next section) with a constant heap usage around 350 MB. In the replay mode (reading 1.6 million tweets from a historic corpus [10]), we were able to parse around 50,000 seconds at 92% CPU usage. This high throughput is possible because we only analyse tweets in detail that match a given topic. Analysing every tweet to find the trending words without further filtering degrades the throughput to about 9,000 tweets per second.

To put this in perspective, there are over 500 million tweets on a typical day or around 8,000 per second, i.e. our sampling is capped at 0.5%. Another interesting issue is how data

ranges change considering time zones. We observed data rates in the USA between 8 AM and 5 PM Eastern Standard Time on a Sunday. As seen in figure 5, there are significantly lower data rates in the morning hours, which points towards the conclusion that the sampling is constant to the overall load.

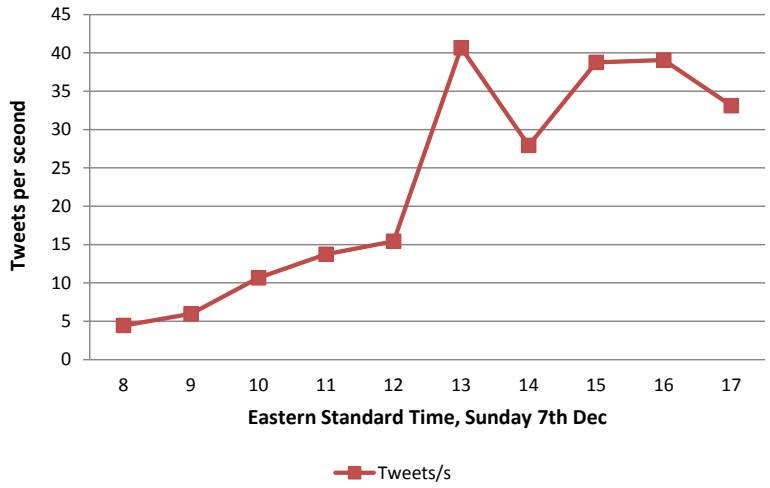

Figure 5: Tweets per second sampled in the USA over different times of the day

5.3 Case study

In this section, we are illustrating a sample use case. The above-mentioned case of the Eric Garner grand jury decision in New York has gained international media coverage and sparked public protests in many cities [12, 8, 13, 14]. It is thus a prime example to investigate how this reflects into live tweeting.

Location	Tweets	Matching Topic	Matches/Tweets
New York	302893	276	0.000911213
London	78897	7	8.87233E-05
Los Angeles	147728	117	0.000791996
Montreal	36281	30	0.000826879
Miami	73724	38	0.000515436
Houston	160477	11	0.000660531
Total	800000	479	0.00059875

Table 1: Sparse reflection on twitter on the topic of the grand jury decision regarding Eric Garner's death.

Table 1 shows the results of capturing 800,000 tweets on this topic (#ericgarner, #icantbreathe, #alivewhileblack) between the 3rd and 7th December. Filtering bounding boxes were around the USA with 5 major regions as well as the UK with all UK tweets clustered towards London. Surprisingly, even though there was major news coverage and demonstrations in New York City, the tweets matching this topic (i.e. the major trending hashtags) only amounted for 500 out of 800,000 tweets. Once again, it is hard to draw conclusions given the fact that the tweets from the streaming API only make up for a small percentage of the real load. For illustration, we display the most popular popular words discussed in this particular topic in form of a tag cloud (see figure 6).

For comparison, we also consider a topic that is not tied to a specific event, e.g. the hashtag "#ukip" in the UK, as seen in table 2. Data rates were much lower in the UK (around 5

Figure 6: Tag cloud of most popular words on the topic of the Eric Garner decision.

tweets/second between noon and 8 PM). Interestingly, the total matches were in the same order of magnitude as in the previous experiment.

Location	Tweets	Matching Topic	Matches/Tweets
Liverpool	30912	12	0.000388199
London	25320	9	0.00035545
Edinburgh	14551	1	6.87238E-05
Cambridge	8078	3	0.000371379
Oxford	21139	5	0.00023653
Total	100000	30	0.0003

Table 2: On the topic of "#ukip".

With such sparse results, it seems hard to gain any insight into the twitter sentiment on a given topic in a reasonable timespan, which is also a result, although a disappointing one. The most relevant positive and negative words are shown in table 3.

#ukip	
Positive	support, great, vote,@davidjo52951945
Negative	crap, accuse, opposition, condemned, cons, lost, sleazy, cold, wild, cave

Table 3: Relevant words in "#ukip". Note that the word " @davidjo52951945" is mistakenly identified as a positive word due to the false positive rate of the Bloom filter used for the membership query.

6 Conclusion and future work

In this project, we have seen how we can combine Apache Storm with probabilistic data structures to achieve efficient stateful processing at the cost of small error rates that we can consider to be negligible when capturing overall trends. To evaluate the system at scale, access to the firehose mode would be necessary. For the limitations of the streaming API, a single laptop computer was sufficient to capture all live data. Surprisingly, even on a trending current issue, the total impact on twitter was barely noticeable. This is specifically disappointing because as a further goal, it would be interesting to analyse more local/hierarchical relationships between locations and events, which would require much more topic matches.

Furthermore, tweets matching a certain topic could certainly be analysed in more detail. Research on sentiment has been employing different machine learning (e.g. Naive Bayes, Maximum Entropy, SVM) as well as linguistic approaches to classify tweets [1, 11, 20].

References

[1] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, LSM '11, pages 30–38, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[2] BBC. BCC Breaking News. https://twitter.com/bbcbreaking, November 2014.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[4] Jeffrey Breen. Github project on sentiment analysis containing classified wordlists. https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/tree/master/data/opinion-lexicon-English.

[5] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, April 2005.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[7] Lauren Dugan. Blog article on average wordlength. http://goo.gl/iWNjrA.

[8] Ryan Gajewski. Eric garner decision: Protesters march on hollywood blvd. http://www.hollywoodreporter.com/news/eric-garner-decision-protesters-march-754678.

[9] Alexander F. Gelbukh and Grigori Sidorov. Zipf and heaps laws' coefficients depend on language. In *Proceedings of the Second International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '01, pages 332–335, London, UK, UK, 2001. Springer-Verlag.

[10] Alec Go. Twitter corpus data. http://help.sentiment140.com/for-students.

[11] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *Processing*, pages 1–6, 2009.

[12] J. David Goodman. Wave of protests after grand jury does not indict officer in eric garner chokehold case. http://www.nytimes.com/2014/12/04/nyregion/grand-jury-said-to-bring-no-charges-in-staten-island-chokehold-death-of-eric-garner.html.

[13] Oliver Laughland. Report on eric garner protests by the guardian. http://www.theguardian.com/us-news/2014/dec/04/we-cant-breathe-eric-garner-protesters-chant-last-words.

[14] Paul Lewis. Obama under pressure over response to police killings after eric garner decision. http://www.theguardian.com/us-news/2014/dec/04/obama-police-killings-eric-garner-decision.

[15] Paul Lewis. Report on eric garner investigation. http://www.theguardian.com/us-news/2014/dec/03/justice-department-eric-holder-eric-garner-investigation.

[16] Nathan Marz. Trident. https://blog.twitter.com/2012/trident-a-high-level-abstraction-for-realtime-computation, August 2012.

[17] MongoDB, Inc. MongoDB. http://www.mongodb.org/.

[18] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 439–455, New York, NY, USA, 2013. ACM.

[19] Philip Howard et al. Opening closed regimes: What was the role of social media during the arab spring? http://pitpi.org/?p=1051.

[20] Michael Speriosu, Nikita Sudan, Sid Upadhyay, and Jason Baldridge. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First Workshop on Unsupervised Learning in NLP*, EMNLP '11, pages 53–63, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[21] The Apache Software Foundation. Apache Hadoop. http://hadoop.apache.org/.

[22] The Apache Software Foundation. Apache S4. http://incubator.apache.org/s4/.

[23] The Apache Software Foundation. Apache Samza. http://samza.incubator.apache.org/.

[24] The Apache Software Foundation. Apache Storm. https://storm.apache.org/.

[25] The Apache Software Foundation. Apache Zookeper. http://zookeeper.apache.org/.

[26] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156, New York, NY, USA, 2014. ACM.

[27] Twitter. Twitter streaming API documentation. https://dev.twitter.com/streaming/overview.