Real-Time Storage and Processing of All Geo-located Tweets in MongoDB

Ali Hürriyetoglu* **Manos Tsagkias**** **Antal van den Bosch***
*** Center for Language Studies, Radboud University Nijmegen**
**** University of Amsterdam**

Twitter has become the source of a huge user-generated content stream of short messages, known as tweets. The content of a tweet text may vary from personal status updates to advertisements. Rich metadata is readily available as well, or may be computed automatically, such as retweet counts (the number of times the tweet was re-posted by other users), the mentioning of named entities, self-provided user information on his or her location, and the geo-location (GPS coordinates) of where the tweet was posted, if the user allows that information to be shared. Analyzing this metadata can enable us to understand the dynamics of information flow and user behavior better than analyzing just the textual content of tweets. In order to benefit from this data, we need to access and process this stream as complete as possible. However, Twitter imposes strict limitations on the access of tweets. It is possible to crawl a random sample of the stream which does not exceed 1% of the whole stream.

In order to overcome these limitations we store a subset of the tweet stream in a compact and flexible environment to be able to analyze it. We aim to get a geographically metadated sample of tweet stream with this method. We present an experiment in which we store as many as possible geo-tagged tweets in a MongoDB database in real time, and in which we count the tweets per geo-box in an incremental map-reduce fashion. Despite free access being limited, it does allow us to harvest the a set of geo-tagged tweets. We access the database from a Python programming environment.

The stream is summarized in spatio-temporal data series which are updated and stored in the database for every geo-box in which at least one tweet was posted during the last five minutes. Although it is possible to calculate these series without storing tweets in the database, we need to store them to be able to change the box and time period size afterwards. This strategy enables us to change the resolution and dimensions of the analysis per region, time period, user etc . easily.

Geo-tagged tweets constitute 1% to 2% of the whole Twitter stream. However, we can crawl up to 1% of it due to Twitter streaming API restrictions. Harvesting this subset yields around 10 million tweets, or 3 GB of data per day. However, we do not need to store tweet attributes that have default values such as null, 0, empty list, and duplicate information. The system infers that an attribute should have its default value in case this attribute is not found in a stored tweet document. This straightforward method results in about 2/3 space gain without losing any information. This is an advantage of using a NOSQL database.

There were some issues with representing time in MongoDB and Python. We needed to convert the tweet creation time to a `datetime' format to be able to do temporal comparisons. Incremental map-reduce uses UTC-based time to process the last five minutes of the stream. Using default current time as provided by Python may introduce errors in comparisons. Moreover, it is crucial to know that month numbers start from zero in MongoDB. The standard geo-indexing capabilities of the database makes it highly suitable for geo-location-based queries.

We analyzed the collection for tweets generated in the geographical scope of Syria for a week, as we are interested in monitoring events as they unfold in volatile areas We have found that although the geolocated tweets are quite sparse, they are representative of many other social platforms, as the tweets include references to Foursquare, Path, Here, Youtube, and Instagram. The geo-tagged tweets furthermore contain many topics, a large range of hashtags, and have a wide balanced geographical distribution.

Further studies will focus on using new geo-indexing capabilities of the database such as geoSphere2D, linking the sparse data, assessing quality of the sample, and scaling the database to larger amounts of data.