Streaming Big Data with Storm

Peter Prescott (201442927)

Introduction

Storm is a Java program designed to reliably process unbounded streams of data. In this assignment we are required to create a Storm topology that performs a simple sentiment analysis on a stream of tweets relating to the coronavirus vaccine.

Middleware Configuration

While Storm can be run as a single local instance, its distributed design is intended to take advantage of a networked cluster of multiple computers. We can simulate this by creating a local standalone cluster.

A Storm program is run by a master *Nimbus* node that delegates tasks to some number of *Supervisor* nodes, which are coordinated by *Zookeeper*. Once this has been successfully set up, we can run the **`storm ui`** command, which allows us to monitor the status of the cluster in a browser at **localhost:8080** (Figure 1) We can then submit our topology to run on the cluster: **`storm jar topology.jar piprescott.Topology`**.

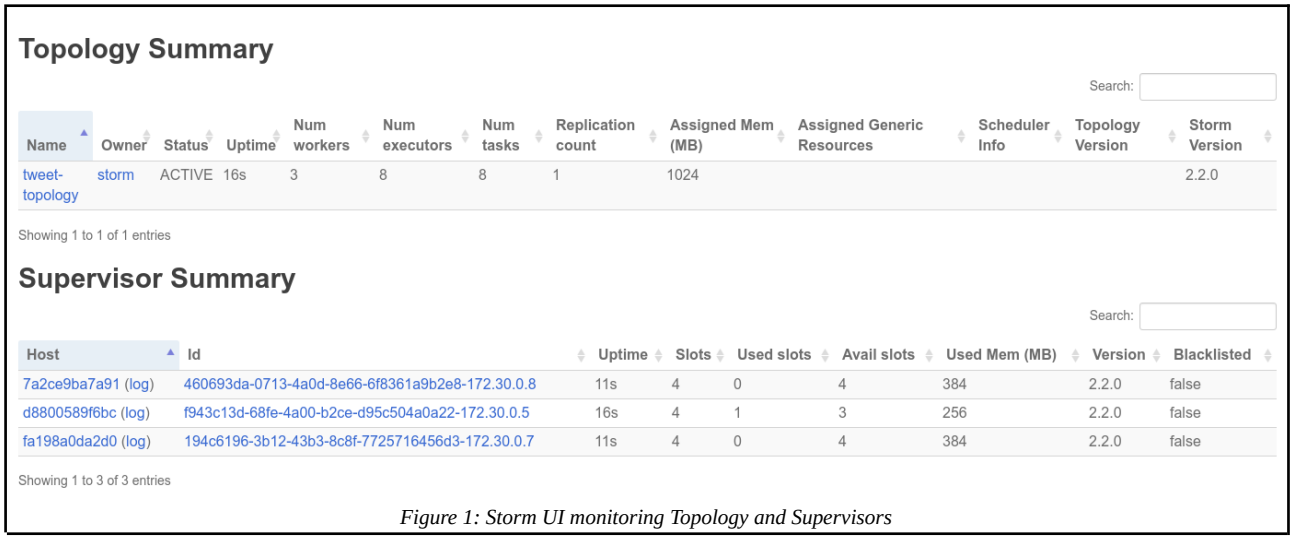

Figure 1: Storm UI monitoring Topology and Supervisors

To simplify the process of setting up these components, and to keep them separate in isolated containers, I described the necessary configuration in a `docker-compose.yml` file.

The cluster can then be deployed simply by running **`docker-compose up`**

(see Figure 2).

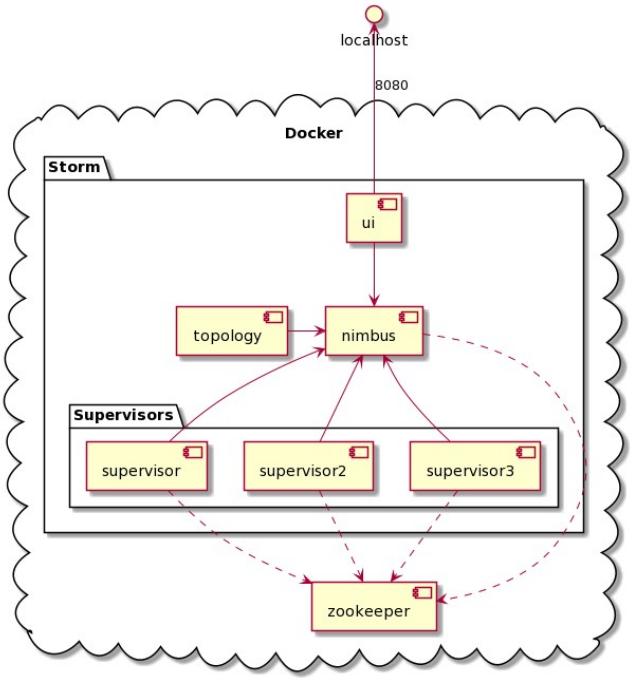

Figure 2: Dockerized Storm Cluster

Data Analytic Design

To design our topology, first we define a `TwitterSpout` class (an extension of Storm's BaseRichSpout). This establishes a connection to the Twitter API with the necessary developer access keys, using the `twitter4j` library. It then listens for, and emits a continuous stream

of new Twitter status messages related to the coronavirus vaccine. We do this by tracking tweets containing "#COVID19", "#COVIDVaccine", or "Vaccine".

We then define a `FilterBolt` class which converts the text of each tweet into a list of lowercase, and then removes any which match a list of common irrelevant words.

We then define a `NegativeBolt` class which calculates a negativity score for the filtered words, by counting the words which match a specified list of negative words. Similarly our `PositiveBolt` class does the same, but for a list of positive words. We then define a `ScoreBolt` class, which classifies each tweet as positive, negative, or neutral, depending on whether the tweet's positivity score is greater than, less than, or equal to its negativity score. It then writes the tweet and its classification to a file, together with the running totals of positive and negative tweets.

We then define our main `Topology` class, which builds our Storm topology by initiating a new instance of `TopologyBuilder()`, on which we set the Spout and Bolts that we have defined.

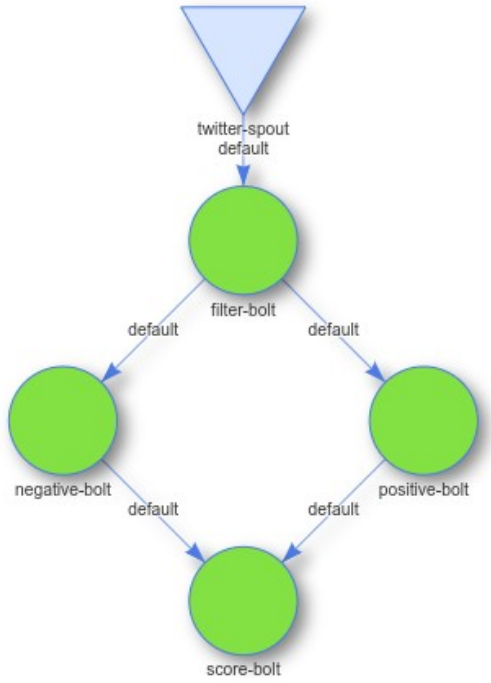

Figure 3: Parallelized Storm Topology

As can be seen in Figure 3, we connect the `negative-bolt` and `positive-bolt` in parallel; the rest has to be ordered sequentially. We could further parallelize our topology by creating multiple instances of the bolts, but for the sake of simplicity we have not done this.

Discussion of Results

To understand how our sentiment analysis functions, it is worth considering in detail a particular tweet: *"Awful glad vaccine is coming at last! #COVID19"*. Our `twitter-spout` passes it to the `filter-bolt`, which converts it to the list `{"awful", "glad", "vaccine", "is", "coming", "at", "last", "covid19"}`, and filters out the common words `"is"` and `"at"`. Then the filtered list of words is passed in parallel to negative-bolt and positive-bolt, which each give it a score of 1 (for `"awful"` and `"glad"` respectively). Our `score-bolt` then finds that the tweets negativity and positivity scores are equal, so classifies it as neutral. This is clearly incorrect, for `"awful glad"` is in fact thoroughly positive -- so we must be aware of the limitations of our very simplistic sentiment analysis.

Nevertheless, we can run our topology, and in doing so I find that after 71 seconds, it has analyzed 137 tweets, classifying 17 as positive, 41 as negative, and 79 as neutral.

Conclusions and Recommendations

In conclusion, we have demonstrated here how to build a simple topology using Storm to analyze the sentiments of a real-time stream of tweets on a subject of immediate relevance.

The obvious recommendation to improve our analysis would be to implement a more sophisticated system of natural language processing to analyze the positive and negative sentiments of a tweet.

We would also be able to analyze more tweets more quickly with greater computational power, and since we have defined our cluster configuration in Docker it should be easily transferable to the cloud.