Yet Another Twitter Sentiment Analyzer (YATSA)

UC Berkeley MIDS Final Project for W251 - Scaling Up

Git Repo | Presentation Slides

Paul Durkin, Ye (Kevin) Pang, Laura Williams, Walt Burge, Matt Proetsch

OVERVIEW

Every second, on average, around 6,000 tweets are tweeted on Twitter, which corresponds to over 350,000 tweets sent per minute, 500 million tweets per day and approximately 200 billion tweets per year. With all this activity, Twitter creates an estimate of 8TB of data per day. The ability to process and analyze data in motion can help a range of industries understand questions like when customer experience is degraded, real-time fraud detection and so on. Our project explores how to set up an end-to-end solution to process tweets in real-time and leverage machine learning and natural language processing to calculate sentiments on any theme using a list of hashtags.

GOALS

- Provision a robust infrastructure to ingest the twitter tweet stream
- Efficiently process and store processed tweets
- Generate aggregated sentiment on any interested theme
- Setup a real-time dashboard with insightful visualizations

TABLE OF CONTENTS

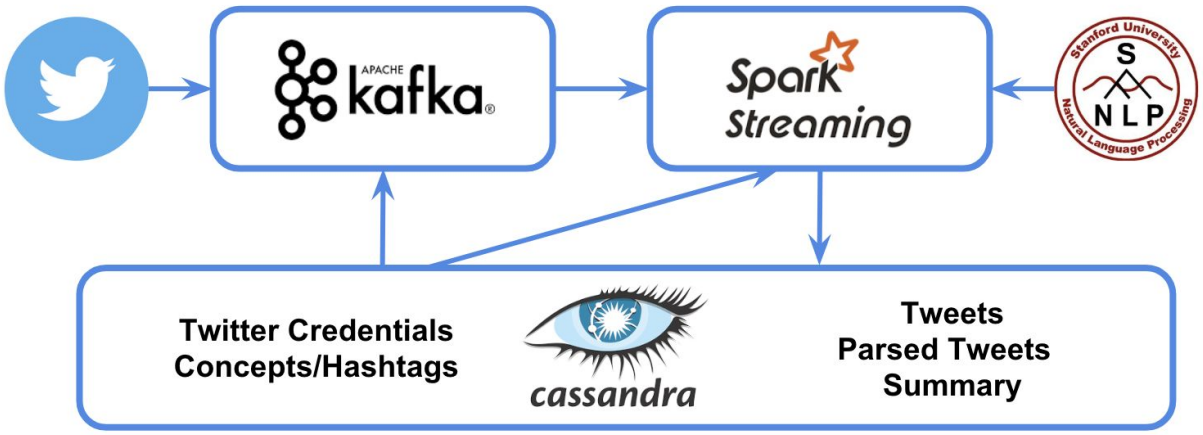
APPLICATION

Data Processing

Overview

We ingest tweets from Twitter API via Kafka, then perform stream processing in Spark, and sentiment analysis over REST API calls to Stanford CoreNLP server before sending processed results to Cassandra to be displayed in Tableau. Raw tweets are persisted in Cassandra as well.

Data Flow



We start by manually populating two Cassandra tables, one with Twitter credentials and another with concepts, or themes, and a list of associated hashtags with for each concept. A python script queries the Cassandra database to collect the Twitter credentials and the full list of hashtags against which to filter the Twitter stream. The Kafka producer creates a message stream of filtered tweets for Spark to consume and additionally saves all raw tweets in the Cassandra database.

A second python script initiates Spark Streaming, which connects to the Kafka producer to consume the filtered twitter stream. We use Pyspark inside this second python script to create two Spark DStreams. The first DStream parses the tweets, manages encoding and data types, sends tweet text to the Stanford NLP server for a sentiment score, and saves a collection of parsed tweet data, including tweet text, hashtag list, sentiment score, and tweet metadata to the Cassandra database in a table titled, “Sentiment.”

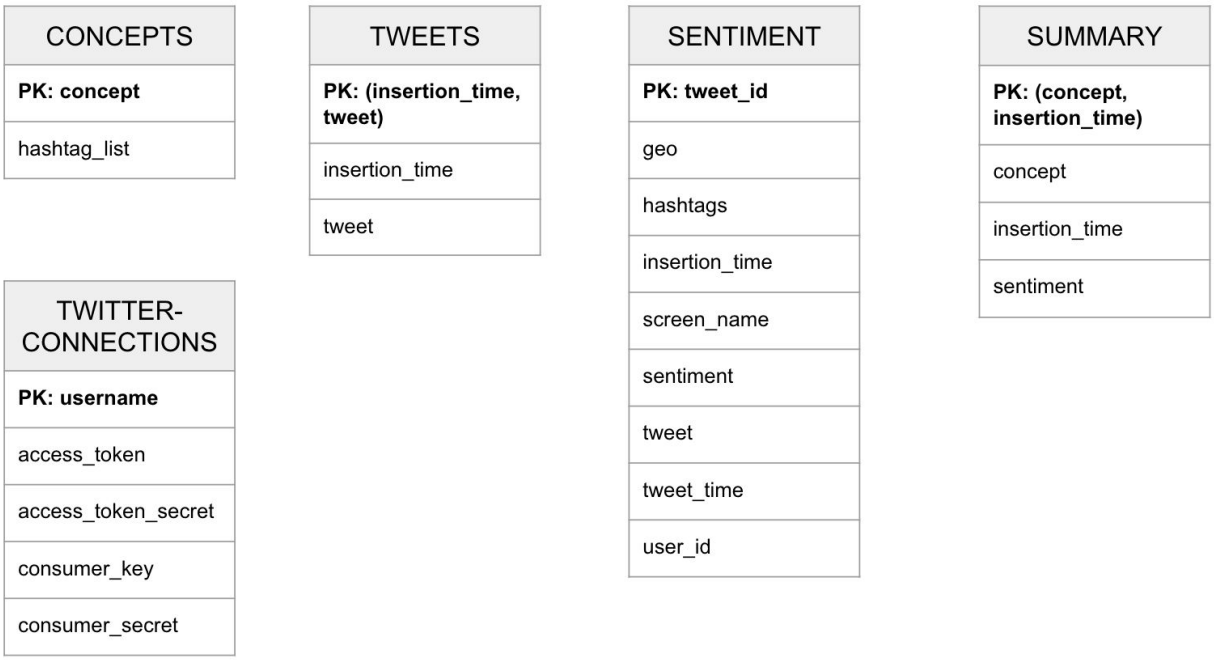
The Spark Streaming process also collects the list of concepts and related hashtags from Cassandra. The second DStream matches a concept to each tweet based on the hashtags in the tweet, using the concept/hashtag data collected from Cassandra, then obtains a sentiment score for each tweet using the Stanford NLP server. This DStream aggregates sentiment scores per concept per window, and saves this data per window in the Cassandra database, in a table titled, “Summary.” The window size is configurable and defaults to 1 minute. This data is displayed in Tableau and is intended to show real-time sentiment trends for any given concept or theme, as specified initially in the Cassandra database.

Utilities

We have a suite of utility scripts to help us with common database operations such as create, drop, and delete tables. In addition, we have setup scripts for Cassandra and templates for seeding tables. Twitter API credentials are also set and stored within Cassandra.

Cassandra Entity Diagram

While Cassandra may be queried by any field in standalone applications, Cassandra must be queried only by primary key in a cluster environment. In some tables, this requires compound primary keys. Tables in Cassandra don't interact with each other, rather they are called by python scripts for use with Kafka and Spark.



Sentiment Analysis

The Stanford CoreNLP server is a freely-available, self-contained software package providing a variety of Natural Language Processing services. One of these services is Sentiment Analysis, and we use its REST

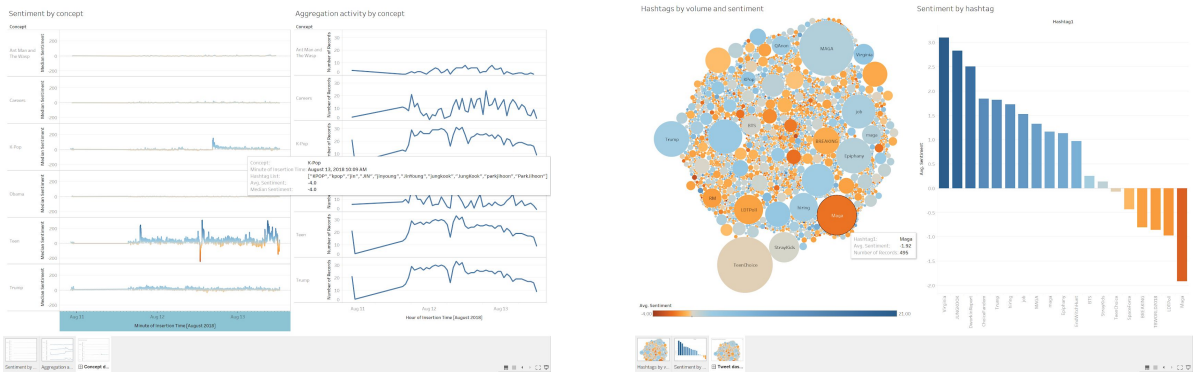
API to measure the sentiment associated with each incoming tweet. The CoreNLP server also supports extensible language models that allow for foreign languages.

The Spark stream is configured to work with the CoreNLP server to retrieve sentiment values between -3 and 3 for each tweet, with -3 being the most negative sentiment and 3 being the most positive. The sentiment value for each tweet is saved to Cassandra, and the aggregated sentiment value per window in the stream is calculated by adding up all sentiment values per tweet for each topic. Thus we are representing mixed positive and negative sentiment as a neutral value, and a high volume of tweets with a strong sentiment in one direction or another during a given time window is represented with a higher aggregate sentiment value.

In the end, the “Sentiment” table in Cassandra saves a single sentiment value per tweet, and the “Summary” table saves an aggregated sentiment value per concept, per a given time window.

Visualization

Demoed during class. Minute-by-minute means of sentiment by concept as well as volume and characteristics of individual tweets are graphed.



Concept- and tweet-level dashboards show concept sentiment and activity atleft. Concept hashtags appear on mouseover for each concept Volume and sentiment by tweet are shown at right.

ARCHITECTURE

Overview

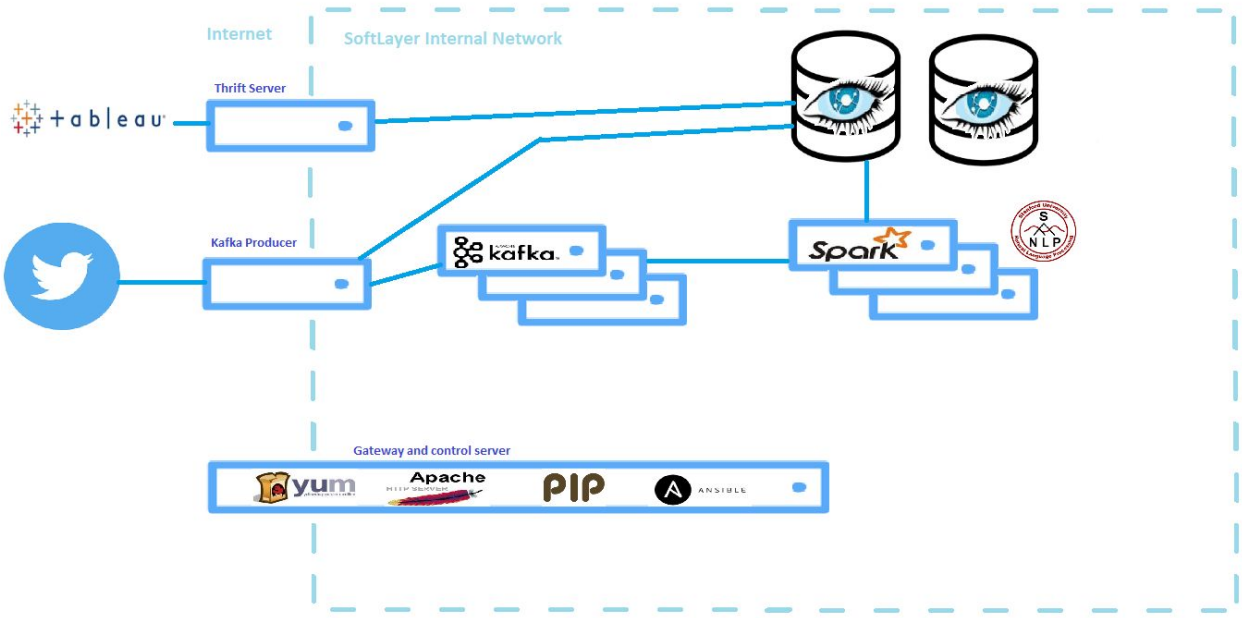


Figure 1: Architecture Overview

This section covers the tools, technologies, and environment used to implement YATSA. First, we look at the two different types of tools, the ones that are employed directly by the solution to process, analyze and store the tweets, and the ones that support the installation and orchestration. The former are Apache Kafka, Apache Spark, Apache Cassandra, Apache Thrift, Stanford NLP Server and Tableau, while the latter is mostly Ansible with several supporting services. After that, we will cover the environment within which all of these were deployed.

Tools used by the solution

Apache Kafka

Apache Kafka is a stream processing system that allows us to collect the incoming stream from Twitter and make it available to our Apache Spark application. It is highly available, scalable and allows both real-time processing of messages (tweets) as well as replaying them.

In our installation, we deploy a single zookeeper server and several broker servers which can be added or removed as demand requires.

We use Kafka to connect to the Twitter API and get a constant stream of tweets that match the criteria we specified. Kafka offers decoupling and buffering for performance out of the box. The decoupling also allows us to perform maintenance on the cluster without any downtime impact to the data stream.

Another key feature of Kafka is the ability to playback the stream according to the retention settings since it saves data to disc.

Spark

We use Spark because it is a fast and general engine for large-scale data processing. Spark introduced the concept of batch processing into streaming by breaking the stream down into a continuous series of micro-batches, which could then be manipulated using the Apache Spark API. Spark also uses in-memory processing, which is what makes it capable of delivering real-time analytics at lightning speed (100x faster than Hadoop).

Our deployment of Spark uses a single master server and a scalable number of slaves (worker servers). Also since we have only one application running, we use the standalone mode which allows our job to use all the resources of the cluster. We have enabled dynamic allocation and the external shuffle service on all nodes so that workers can be added and removed dynamically.

Cassandra

Cassandra can be always on, even during massive hardware and network failures by utilizing a design first widely discussed in the Dynamo paper from Amazon. Cassandra is typically classified as an AP system, meaning that availability and partition tolerance are generally considered to be more important than consistency in Cassandra.

We picked Cassandra because we value those two qualities over consistency and set our replication to 3. We also deploy a minimum of 3 nodes and scale up and down from that base.

Stanford NLP

The Stanford NLP server (detailed earlier) is a service which runs as a separate process on each Spark node, providing a REST API. The Spark YATSA workers communicate with this to get a sentiment score.

Apache Thrift

Apache Thrift is the technology underlying a middleware layer handling communication between BI tools like Tableau and our backing datastore written for Cassandra. The Thrift server exposes the data from tables known to the Hive metastore to external clients such as Tableau. When a SQL query is sent to the Thrift server requesting data from a table in the metastore, a Spark job is launched to query the backing datastore to return the data to the client.

Tableau

Tableau helps us see and understand the data. Tableau is a killer for creating visual dashboards – easy, fast, and self-service.

Servers, Network and Security

Our infrastructure is built entirely in IBM's SoftLayer cloud, using the latest Centos 7 virtual servers. We deliberately used low powered VMs as we wanted to implement clusters and test our ability to scale out resources.

IBMs SoftLayer Cloud provides two networks, one private and one public. The private VLAN is multi-tenant aware and so includes isolation from the traffic of other subscribers. The public network uses routable public IP addresses with no restrictions on incoming traffic.

While firewalls are available from SoftLayer, (they come at an extra cost) we did not use these. Instead, we considered several other hardening options to support the security of the deployment.

Firstly, because the private VLAN provides isolation between tenants we rely on this for protecting many of our VMs. Most of the servers are only deployed on this VLAN and have no access to the internet. Since these can only be accessed by other servers in the same VLAN, we did not protect these with firewalls.

Secondly, in all cases, we allow only key-based access to the servers and explicitly configured this in our sshd configuration file. This means that password crackers will not help to compromise the servers.

Finally, all servers that require an internet connection have a very tightly controlled internal firewall. These servers are:

1. Gateway server: our administration portal to the internet. Only allows ssh in, with key-based authentication
2. Kafka producer servers: connect to Twitter to pull in tweets. This does not allow any incoming connections on the public interface, and so no logins are possible. We initiate the connection to Twitter from the inside, so it is not blocked, but all incoming connection requests (TCP/IP SYN packets) are dropped.
3. Thrift server: this interfaces between Tableau and Cassandra. Again this is key-based login, and the remote hosts that will connect are explicitly configured on the firewall. All other external access is denied.

Installation and configuration

Installation and configuration of the virtual machines were performed with a combination of SoftLayer CLI and Ansible. Because the majority of the VMs have no internet access, Ansible is supported by an internally deployed Yum EPEL repository, a Python PIP repository and a software repository, all made available through Apache Httpd.

Ansible is a tool for configuring provisioned servers using Playbooks. We created the playbooks detailed in Table 1 to allow us to add or remove nodes as required.

Playbook	Description
Cassandra	Adds a new node to the Cassandra cluster
Cassandra Decommission	Removes a node from the Cassandra cluster, properly disconnecting it first

Decommission	Removes a node that is no longer necessary, should only be used for nodes that do not require a process for shutdown and removal
Kafka Producer	Sets up a producer. This server is public facing so the playbook also sets up secure firewall
Kafka	Sets up a new Kafka broker and joins it to the existing cluster
Spark Master	Sets up the one Spark master. Includes deployment and start up of the YATSA job
Spark Slave	Sets up a new Spark slave and joins it to the master cluster
Spark Decommission	Decommissions a spark worker so that the running job is not impacted
Thrift	Sets up a new thrift server
Zookeeper	Sets up a Kafka server with zookeeper and a broker

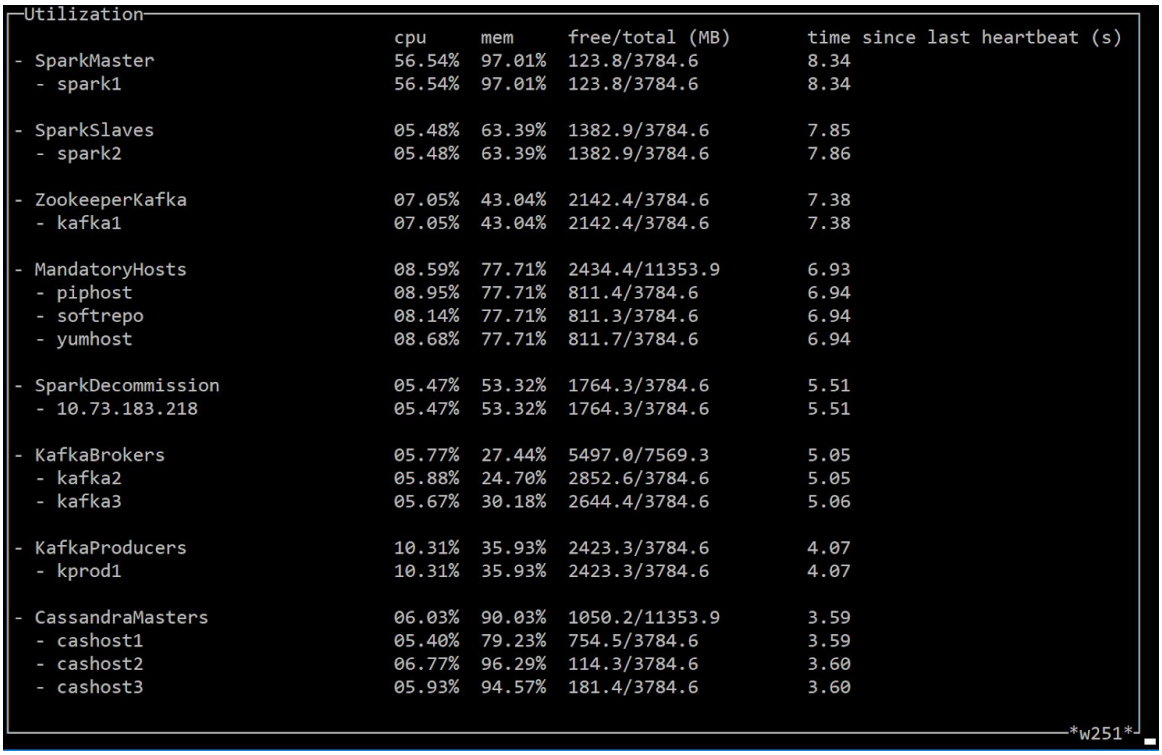
Table 1: Ansible Playbooks

As an alternative to implementing a DNS, we created a role (a subcomponent of a Playbook) that updates the /etc/hosts files on all servers when changes are made to the network. This, along with using alias names for services allows us to be independent of the hostnames and IP addresses of the VMs.

We had one exception to building all the components with Ansible, and that was Cassandra. While we could have provisioned the initial Cassandra nodes, it would have meant wiping the database each time we reprovisioned, and so instead we decided to implement a playbook to scale up and down the nodes in the Cassandra cluster.

Monitoring

While there are sophisticated monitoring options available, we elected to use a simpler approach. The volume of data coming from the Twitter “garden hose” is not expected vary significantly over time, so monitoring the health of individual nodes was prioritized over allowing a tool-free reign to spin up as many nodes as it deems necessary. The tool we developed will enable developers to monitor cluster health by node type as well as the health of individual nodes. As the number of tracked hashtags grows, cluster load is expected to increase nonuniformly, so deciding when to spin up new nodes using Ansible to meet demand and to monitor unhealthy nodes becomes paramount.



CHALLENGES

There were significant challenges in resolving version incompatibilities between Scala, Pyspark, Cassandra and several different spark-python-cassandra connectors, all of which required different methods, dependencies, and compatibility requirements to transfer data seamlessly between Spark and Cassandra. These issues were particularly challenging to address due to lack of reliable documentation about required versions for these different layers, as well the fact that the pyspark_cassandra connector, the package we eventually chose to use, has been managed by various organizations for different versions.

Finally, several challenges resulted in coding errors that required Scala knowledge. We were grateful for the introduction we had to Scala in our earlier work in the course.

FURTHER IMPROVEMENTS

Elasticity

If we had more time and the ability to work with Kubernetes, we would like to test its native elasticity capability while using the same stack of services we defined and set up. While we experimented with

Terraform for deploying infrastructure as code, we observed that it is most productive when it can be run with an orchestrator as well as a compatible service account to enable parallelism.

Machine Learning

We took the simple approach of scoring the tweet by itself for sentiment analysis. Future iterations could include other metadata and try alternative NLP approaches to improve the predictions.

User Interface Improvements

Our existing user interface reflects data in Cassandra via a Tableau dashboard, updated in real time. Ideally, a user interface would also allow a user to input a concept with related hashtags, and start up another Kafka producer for that concept with an associated spark flow to populate the Cassandra database.

Enterprise Solutions

Our personal Twitter streams limit our analysis to current trending topics and also impose rate limiting for the total number of tweets we can ingest. An enterprise Twitter application account would allow access to historical Twitter data, which we could access via the same streaming process, where the Kafka producer was directed to the historical Twitter data rather than the current trending stream.

Docker

While we are successful in deploying services on VM boxes directly, Docker containers in a Docker Swarm would be an easier solution to manage the many libraries and dependencies. While working with containers add its complexity (especially networking), overall, they better protect against service failures and would create a more fault-tolerant solution.

CONCLUSION

Using the Stanford deep learning CoreNLP, our solution analyzes the sentiment of tweets in real time and persists them to a NoSQL storage. The infrastructure is designed to scale out easily by adding capacity in workers as needed. And because we put Kafka in front of Spark to ingest the data stream, we ensured minimal interruptions and the ability to handle varying volumes of tweets

For our proof of concept, we analyzed tweets related to movies, but the solution could be used for any concept or theme that have a set of hashtags defined for it. For example, a celebrity, global warming, or the legalization of marijuana. We also envision that this solution could benefit field experiments, by measuring the differences in effect on how a message is presented, and so help to optimize the intended outcome. As a next step, we are keen on extending this solution based on the improvements listed above.