Twitter Word Count App Architecture

I. Directory and File Structure

Overview

The root directory contains the below folders and files:

- <Scripts> Directory
- <Document> Directory
- Architecture.docx
- Readme.txt

Scripts Directory

This directory contains the major production codes that Storm runs upon. It has several sub-folders under it which I illustrated further below.

Directory	Description
Key Codes	This directory listed the 6 scripts this exercise requires the student to develop, and I listed them separately in an individual folder mainly for grading purposes. It contains tweetwordcount.clj, tweets.py, parse.py, wordcount.py, finalresult.py and histogram.py.
tweetwordcount	This folder contains all the key and supplemental parts of codes the streamparse would run upon. The key folders or files are: Topologies: contains a clj file that describes the topology in terms of Directed Acyclic Graph (DAG) of Storm components, Bolts and Spouts. Src/spouts: contains tweets.py that serves as a source of streams in a topology. The spouts will read tuples from twitter streaming and emit them into the topology. Scr/bolts: contains parse.py and wordcount.py where all the data processing work is done. Twittercredentials.py: contains the twitter Apps credentials
Finalresult.py	This file contains the codes that query the postgres database, and return a specific type of result given some arguments. This script gets a word as an argument and returns the total number of word occurrences in the stream
Histogram.py	This file contains the codes that query the postgres database, and return a specific type of result given some arguments. This script gets two integers k1,k2 and returns all the words that their total number of occurrences in the stream is more or equal than k1 and less or equal than k2.
Readme.txt	This file contains execution instructions for setting up directories, creating database and tables, and running streamparse etc.

Document Directory

This directory contains a collection of screenshots capture the real-time execution of the application as well as the output results:

- screenshot-extract-results (1).gif contains the sample run of script finalresults.py
- screenshot-extract-results (2).gif contains the sample run of script histogram.py
- screenshot-storm-components.gif captures the output from a real-time execution of the application
- screenshot-twitterStream.gif shows the output when running the hello-stream-twitter.py
- plot.gif plots a bar chart that shows the top 20 words captured in the twitter stream

Architecture.docx

This document is complete documentation of the twitter application including directory and file structure, application idea, description of the architecture, file dependencies, any necessary information to run the application, etc.

Readme.txt

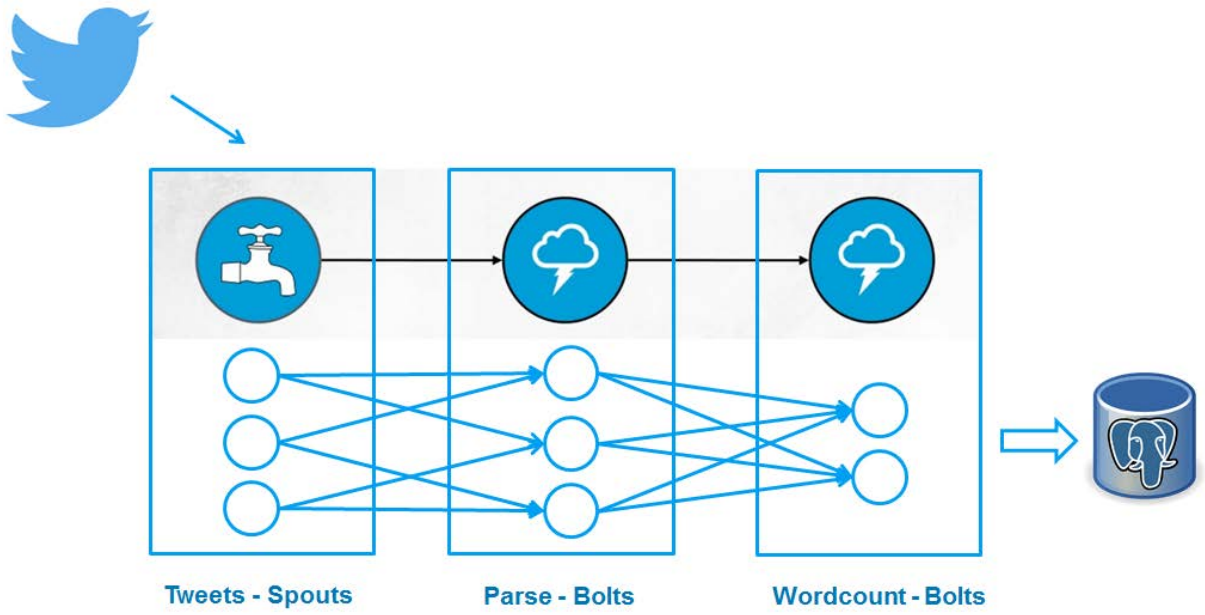
This document contains basic information about the folder structure at root level.

II. Application Idea

The application is built to analyze the word occurrences of the live streaming tweets captured real-time. The application monitors and processes the twitter stream, and stores the information in a backend database. The information gathered can be saved and used for later ad-hoc queries and analysis.

The word occurrence analysis can help to reveal trending words and topics within a period of time, and discover public interests and social behavioral trends. All these valuable information can be leveraged in a variety of interests, for example, ads targeting.

III. Application Architecture



The application is built upon the similar idea of Storm which consists of 3 major pieces in the overall architecture: topology, spout and bolt.

Topology defines the processing pipeline, links the different functional pieces. It explicitly defines the output for each function segment and passes it as an input for the following segment.

Spout is set up in order to capture the real-time twitter stream. By providing the twitter developer credentials, the Spout can monitor the tweets update, captures them and send the tweets to the Bolts for process.

Bolt is built for processing the tweets and converting them into the final format we could use for analysis. In the application, we have two bolts: parse and wordcount. Parse Bolt is used as an intermediate step that split the sentence into a collection of separate words and filters out a series of special character or word that is invalid. The output of the Parse Bolt will be passed onto wordcount Bolt.

Wordcount Bolt interacts with the backend database which is built out of Postgres DB. The Bolt queries the database for the historical occurrence for each single word and update the word count based on the new tweets statistics. If it encounters a new word, it will add one more entry for that word which has an occurrence of 1.

The Storm could only capture the real-time word count during the session while the application is run, therefore, in order to save the word count information permanently for later use, the application save another copy of the info in the backend postgres DB. The business user can get the information by either

query the data from postgres or run the two python helper codes that help to extract the word count for a certain word or generate a histogram for word count based a certain criteria.

IV. File dependencies

Before running the streamparse, please fill in the Twitter developer API credentials in the file of Twittercredentials.py under /tweeetwordcount directory.

When running the streamparse, it requires changing execution directory to /tweeetwordcount. The application will look for topology file under /topologies folder and all spout and bolt files need to sit under /src folder.