Assignment B

Real-time analysis of Twitter Streams using Storm

Weightage: 150 points (15%)

Posted date: Wed 22 Feb, 2017

Due date: Fri 10 Mar, 2017 by 11:59PM IST

Introduction

In this assignment, you will learn to apply distributed stream processing and Apache Storm for analyzing tweet streams. This dataset is the same 1% sample over 2 months as the one for Assignment A, except that this is replayed as a stream for online processing rather than available as a static file for batch processing[1]. You may reuse any open-source parsing libraries for the Twitter objects, with clear attribution in your report and code.

For each of these tasks below, write a ***Storm Topology in Java*** with the given filename. Each file should contain the bolt classes and the topology definition in the same file. You must develop the idea for and write all the core Storm logic code by yourself, without external help. The goal is to help you think in terms of streaming dataflows and develop algorithms.

You are provided the base code for "replaying" the tweets present in HDFS file as a spout. You may modify this code as necessary to generate the streams for the tasks below. Make sure to incrementally pre-fetch data from HDFS (e.g. 10MB at a time) and not load the entire input file and run out of memory.

Also, you will submit a ***report*** that contains the dataflow DAG, and a brief comment on the bolts used in each, plots of the results, and a brief discussion of the observations. Document your code. Unless otherwise stated, you should run your jobs and provide results using at least 10% of the entire dataset – more the better, but given the shared resource, make sure to try on smaller before you use larger data. Clearly mention the dataset range you used and the temporal scaling factor for each task. Equal weightage is given to the source code and the report.

1. Extract, Transform, Load *[25 points]*

Develop a topology to perform pre-processing over the input JSON stream. You will extract the tweet from the JSON, perform NLP pre-processing such as normalizing to the same case, removing stop words, stemming, removing URLs, etc. as a sequence of bolts, extract the hashtags and common emoticons from the tweet, remove punctuations, and perform addition pre-processing necessary to write a CSV row for each tweet to HDFS, of the form:

```
<epoch_time>,<timezone>,<lang><tweet_id>,<user_id>,<followers_count>,<friends_count>,<normaliz
ed_tweet>,<hashtags>,<emoticons>
```

[1] https://dev.twitter.com/overview/api/

The epoch time is the Unix time in seconds, timezone should be between -1200 to +1200, lang is the 2 character language in the JSON, and user and tweet IDs are as present in the JSON. The last three columns should have space separated words in the tweet, hashtags and emoticons, respectively.

You can "replay" the tweets from the JSON input, in the order at which it appears in the sequential JSON input files, at as fast a rate as possible for this task.

Name this file as **`TweetETL.java`**.

2. Realtime Statistics over Tweets *[40 points]*

Compose a topology that takes the CSV tuples as input and generates several real-time statistical summaries and visualization on this. Specifically, generate two statistics as part of your topology:

i) Find the top trending hashtags using a decaying window bolt, as described in the algorithm in Sec 4.7.3 of the Leskovec text book on Mining Massive Datasets.

ii) Maintain the distinct approximate count of the number of unique users and hashtags using a bolt implemented for the Flajolet-Martin algorithm in Sec 4.4.2 of the Leskovec book.

In addition, generate one or more additional real-time statistic from the stream. Ensure that the final bolt generates visualizations periodically (e.g., using Java XChart library), or sends the output stream to a service (e.g. rickshaw.js) that can perform realtime visualization. Develop and use bolts that perform filter, join, aggregation and visualization, as relevant.

You can "replay" the tweets from the CSV file at up to 100x the base rate. Ensure that the spout emits the tweets in strict chronological order within each pre-fetched dataset, sorting in-memory if necessary.

Name this assignment file as **`TweetStats.java`**.

3. Sentiment Analysis on Tweets *[55 points]*

Here, you will be building a topology that performs three forms of sentiment analysis over the real-time streams. For convenience, these can be composed as part of the same stream.

i) Given an input keyword, identify the most relevant tweets for the keyword and return the top 10 of the matches in rank order in each minute. Use statistics like follower and followee count, appearance of the keyword as hashtag, and diversity based on the timezone for your ranking function. Some ideas are present here[2].

ii) Use sentiment analysis to examine the mood of the population (not individual) based on the twitter stream across time. Use the keywords for the 13 emotional categories given in PANDAS-X[3],[4], to associate each tweet with zero or more emotions. Examine how the emotions, accumulated across say 1 minute windows, change across time. E.g. plot X Axis as

[2] Leveraging the Semantics of Tweets for Adaptive Faceted Search on Twitter, Fabian Abel, Ilknur Celik, Geert-Jan Houben, Patrick Siehndel, ISWC 2011

[3] THE PANAS-X Manual for the Positive and Negative Affect Schedule - Expanded Form, David Watson and Lee Anna Clark, The University of Iowa, 1994

[4] ANAS-t: A Pychometric Scale for Measuring Sentiments on Twitter, Pollyanna Goncalves, Fabricio Benevenuto, Meeyoung Cha, SYSTEMS, MAN, AND CYBERNETICS, PART C: APPLICATIONS AND REVIEWS, 2013

the window number, and Y axis as the normalized weight of each emotion. Visualize the output offline.

iii) Use emoticons to find out if a tweet is positive or negative, and emit a 0 or 1 respectively[5]. Use this as a bit vector to measure the mood of the population as a count of the positive tweets over a sliding window of 24 hours, by implementing the Datar-Gionis-Indyk-Motwani Algorithm for sliding window counts, given in Sec 4.6.2 of Leskovec[6].

You can "replay" the tweets from the CSV file at up to 100x the base rate. Ensure that the spout emits the tweets in strict chronological order within each pre-fetched dataset, sorting in-memory if necessary.

Name this assignment file as **`TweetAnalysis.java`**.

4. Scalability *[30 points]*

Does Storm weakly scale with the input stream rate? Demonstrate using any 2 topologies from the above solutions. Plot the data rate/slot count in X Axis, time taken in Y axis. Discuss. (**`ScaleXYZ.java`**, where **`XYZ`** is the file name of the earlier task you are scaling)

Submission Instructions

For all tasks, you should submit the source files integrated with Maven's pom.xml that compiles without error, the log files generated for your job, the list of HDFS output file directories along with the MD5 checksum of its files.

i. Name your source folder as **`username-ds256-beta/`**. Replace "username" you're your cluster account username. Make sure the root of the folder contains the **`pom.xml`**, and an **`output.csv`** CSV file with path of HDFS output files and their checksums, **`log/`** folder containing logs generated for the final output reported, and your assignment report **`username-ds256-beta.pdf`**. Do not include jar or class files in this folder.

ii. Tar your folder into a single file with the filename as below.

```
tar cvf username-ds256-beta-src.tar username-ds256-beta/
```

iii. Calculate its MD5 checksum for the tarred tile

```
md5sum username-ds256-beta-src.tar
```

iv. Zip the tarred file with a *strong password*.

```
zip –e username-ds256-beta-src.tar.zip username-ds256-beta-src.tar
```

Copy the encrypted file to **`~/ds256/submission-beta/`** folder in the head node and email the *password* and *MD5 checksum* to Jayanth by the deadline with the subject line "**`assignment-ds256-beta username`**".

If you upload an unencrypted file to the folder, or you use a weak password, you will get 0 (zero) points.

[5] Go, Alec, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision." CS224N Project Report, Stanford 1.12 (2009).

[6] MAINTAINING STREAM STATISTICS OVER SLIDING WINDOWS, MAYUR DATAR, ARISTIDES GIONIS, PIOTR INDYK, AND RAJEEV MOTWANI, SIAM J. COMPUT., Vol. 31, No. 6, pp. 1794–1813

Rules

- You are working in a shared cluster. So make sure that your source files are kept secure, and are NOT readable by any other student account. Disable group and world read and execute permissions on your home folder. Make sure the jar submitted does NOT contain source files. Do not store source files in /tmp, on HDFS or any other publicly readable locations. Any student who violates these rules will get an automatic 0 (zero) for that assignment.
- Do NOT look into the source code of others, even if others are in violation and source code is "lying around" in some folder. If it is not yours or a shared dataset, do not be curious. If you come across such violations, please bring it to the attention of the TA and Yogesh immediately.
- We will pass the submissions through *plagiarism checks*. If there are noticeable similarities between different submissions, you will get an automatic 0 (zero) for that assignment. Repeat offenders will get grade point reductions or failing grades.

Guidelines

- Given human readable names to your jobs and include your username in the job. This helps identify issues.
- Run your experiments on a subset of the data initially before you run it on the full dataset. Since it may take 10's of minutes or hours to analyze the whole data, limit your full dataset analytics (i.e. all ~4000 files). Instead use a subset of the files for testing and debugging. You can also play around with increasing the temporal scaling rate to replay at a faster rate than the original.
- Watch your jobs for exceptions and errors. An incorrect map or reduce method can appear to run slowly while it generates a lot of exceptions in the background, overflowing the log files. Monitor the status using Storm application status, and tail the log output file watching for exceptions.
- If you notice someone else's application is taking up a lot of resources, is behaving abnormally or causing the cluster to be unstable, bring it to the attention of the TA and Yogesh. Do NOT kill others' job.
- Do NOT submit a large number of topologies at the same time. While the batch system does schedule jobs, do not hog the cluster with many long running jobs. If we find students dominating the cluster, they will be warned initially, and if repeated, their jobs may be involuntarily terminated.
- Think about the number of resource slots you will be using, the number of bolts and their parallelism hint, the selectivity of the bolts, and the number of tuples you will be processing. You should have a reasonable estimate of the quantity of resources you need and the time required.