Dimitar Dimitrov 18-940-080
Svilen Stefanov 18-950-907

Data Stream Processing and Analytics
Semester project
March 20, 2019

Design document

Design choices

- We decided to use Flink because of its good documentation, community support and wide popularity in industry.
- Our application will read input from the provided .csv files. We decided to use Kafka as it is easier to use alongside Flink compared to other alternatives. We will have two threads in Kafka, one reading from the .csv files and putting the data into a message queue and another reading from the queue and sending the data to the Kafka producer. By separating the reading and sending in 2 separate threads we intend to reduce the simulation latency compared to alternative designs.
- Initially, we plan to output our results in simple text files. If all other milestones are completed we can replace the text files with an user friendly web interface.
- We plan on using several external systems. For the static data either a relational or graph database could be helpful for determining similarities between users as described below in the Recommendations section. We have considered trying out a Neo4J or a SQL database for the computation on this static data. Additionally, we are planning to use machine learning libraries for the Unusual activity detection.

Task analysis

In this chapter we discuss the 4 major parts of the semester project that were stated in the description of the task.

Data preparation

We decided to define 3 topics per postID, one for each of the predefined stream .csv files. This would allow our system to scale to multiple machines. To keep the task as realistic as possible, we plan not to preprocess, normalize or modify the data in Kafka in any way and will handle this during the aggregation using Flink. Cleansing might be needed only for some parts of the sentiment analysis but we would first experiment without removing special characters, stemming or handling stop words.
Watermarks will be timeWindow-based, making sure that messages with timestamps are correctly counted and waited upon in a 30 minute time interval.
As described in the project specification, our implementation will take into account messages received out-of-order (simulated by an added random delay) and will be tested using the proposed speedup factor to improve performance.

Active posts statistics

We will use the watermarks generated by Kafka to divide the time into 30 min long time windows. For each window we will first partition the data from the comments and likes streams based on the PostID and send it to all nodes. Each node will then create a per-window hashmap which uses the PostID as a key and stores an object containing all the per-post statistics we are interested in. Example of such statistics are the number of comments, the number of replies and hashmap of unique people that posted comments. We will store 24 such per-window hashmaps in a FIFO -

one for each 30 minutes window in the last 12 hours. When a new window is created the oldest window will be removed from the bottom of the FIFO and the new one will be added to the FIFO's head. When new comments or likes arrive for a post at a node it finds the post in the hashmap at the top of the FIFO and updates the associated counters. When a summary statistics need to be generated for the post at the end of the 30 minute window we will use all hashmaps to figure out which are the active posts and to calculate the required counts. To account for the 1 hour long windows for the unique users query we will also have a hashmap.

Recommendations

For this task, we would pick 10 random persons in the beginning and compute a list of their top-5 most similar persons according to a similarity measure that we will define.

We are yet to design the exact formula for our similarity metric with its proper weights and normalization based on research and results from some empirical experiments that we would like to conduct. However, it is going to be composed of the following statistics:

- static data - considers meta-data like location, spoken languages, organizations etc.
- posts or comments of people that you liked or likes/comments of your posts - direct indication of similarity of view points or similar tastes. Here we will distinguish between direct and indirect liking. An example of an indirect like would be a post of a user a comment of which you liked.
- people from the same forum - signifies similar interests in a particular field
- people that commented/liked the same post - we expect this to be the most challenging metric to implement given the stream memory constraints

These were picked based on our understanding of their relevance for the user-to-user similarity and our judgment about their implementation plausibility and performance compared to other possible parameters.

Unusual activity detection

There are two types of unusual activities we distinguish. One is spam posts or comments. The other is unusual behaviour for a particular user, which can occur when the user has his account stolen. For the spam posts/comments we will extract the content and split it into words or sentences. We will then apply some machine learning on these tokens to determine if the content is unusual. We will look at several possible solutions including using pre-trained Naive Bayes or neural network classifiers and using unsupervised learning solution like online k-means clustering. To apply the clustering we will need to extract features from the text. We will look into using word/sentence embedding extraction for that.

For the stolen account scenario, we plan to use the static data for things like location along with the user's previous posts to determine if a post/comment is genuine. If several suspicious comments are detected we will notify the moderators for further inspection. The information in the posts that can be used to determine if they are unusual includes location from which it was posted and text style of the content. We expect the stolen account scenario to be harder to implement and will be only implemented if time allows.