Spring Term 2019 Dr. Vasiliki Kalavri

Data Stream Processing and Analytics

Semester Project

Problem Statement: In this project you are going to use stream processing tools to build the backend of a hypothetical social network. To start with, you will develop tools to replay and ingest streams of posts and comments. Building upon this, you will then familiarize yourselves with the fundamental streaming operators for grouping, counting and windowing by writing streaming queries to compute aggregate statistics about the overall level of activity in the discussion forum. Thereafter, you will engage your creativity and extend the data pipeline to continuously suggest recommended friends and detect people with suspicious behaviour. In this latter open-ended part, you will experiment with and analyze different metrics to rank users by similarity and identify anomalous behaviour using online clustering.

General Remarks: To successfully complete the tasks in this project, you will have to use concepts and techniques introduced during the lectures and hands-on exercises, as well as individual research and critical thinking. Some of the tasks described below are intentionally designed to allow a certain degree of creative freedom in their implementation. You and your partner will have to make a choice of tools, architecture, and algorithms to realize the project; there is no single right answer!

The objective is to evaluate your ability to:

- understand design alternatives in terms of their features and limitations,
- argue about your design choices,
- translate these choices into effective streaming application code.

This project constitutes **40% of your total course grade** and will be carried out throughout the semester in **teams of 2 students**. All tasks will be carried out in collaboration with your partner except for **the final report** which **will be submitted individually**.

The project must be implemented using one the following stream processing tools:

- Apache Flink: https://flink.apache.org/
- Timely Dataflow: https://github.com/TimelyDataflow/timely-dataflow

Deliverables

The project deliverables consist of:

- three (3) written reports as described later in this document (**30 points**),
- the full code implementing the analytics tasks, as well as any auxiliary code for data pre-processing and formatting (**70 points**),

The respective projects milestones are spread out over the semester as follows:

Deliverable	**Deadline**	**Where?**
Team registration	Friday, 1st March 2019	Email
Design document	Monday, 18th March 2019	Moodle
Midterm progress report	Monday, 29th April 2019	Moodle
Full code implementing analytics tasks	Sunday, 26th May 2019	Gitlab
Final project report (individually)	Saturday, 1st June 2019	Moodle

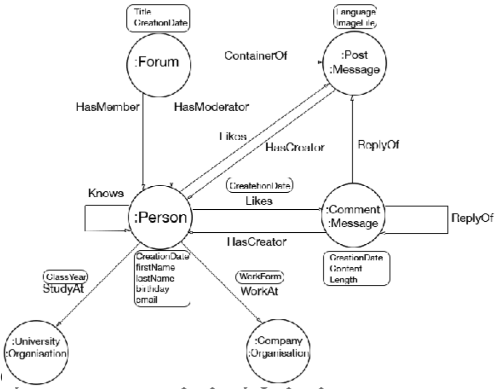
Unless mentioned otherwise, each project milestone should be uploaded by at latest midnight (23:59) on the day of the respective deadline at the following URL:

https://moodle-app2.let.ethz.ch/course/view.php?id=10535

Dataset

You will use synthetic social network data from a hypothetical chat room. The dataset mimics the operation of an online forum where people share messages containing text and images and interact by replying, leaving comments and upvoting posts with likes. People form smaller sub-groups by using tags to group their messages into topics. The activity within a community establishes friendship relations and people can explore the network in search of others with similar interests or located at the same geographical location.

An abstract model of the most important entities and relationships is shown in the figure below[1]:



[1] Taken from Steer, Benjamin A., et al. " nigration." *Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems*. ACM, 2017.

The data has been generated using the LDBC Social Network Benchmark DATAGEN[2]. A more thorough explanation of the data schema and all fields can be found in the following references:

- CSV formatting: https://github.com/ldbc-dev/ldbc_snb_datagen_deprecated2015/wiki/Generated-CSV-Files

- Benchmark specification (see Section 2.3, Page 13): https://ldbc.github.io/ldbc_snb_docs/ldbc-snb-specification.pdf

The following two datasets are provided for download:

- Small (1K people): https://polybox.ethz.ch/index.php/s/qRlRpFhoPtdO6bR
- Medium (10K people): https://polybox.ethz.ch/index.php/s/8JRHOc3fICXtqzN

The data is organized into two folders. The **streams** folder contains time-ordered streaming data that represent events while the **tables** folder contains static data describing properties, such as user locations, workplace, and interests.

For your convenience, some of data has been pre-processed, sorted by timestamp, and combined to create the following data streams you can easily feed to your analysis pipelines. The streams folder contains the following streams:

- **post_event_stream.csv**: a stream of post publication events enriched with information about the post creator, tags, forum ID, and place ID of the post.

- **comment_event_stream.csv**: a stream of comment publication events enriched with the comment author, information about whether the comment is a reply to a post or another comment, and the place ID.

- **likes_event_stream.csv**: a time-ordered stream of like actions.

The files have been generated using an *old* version of the LDBC benchmark as provided by the SIGMOD 2014 Programming Contest[3]. If you would rather generate fresh data yourself, you can do so using the LDBC-SNB data generator[1]. In that case, note that data schema might differ between the old and the current version of the benchmark and that you will need to apply the pre-processing on your own.

Analysis Tasks (70 points)

#0. Data preparation – 10 points

As a first task, you should build one or more re-playable, timestamp-based data sources that produce the data streams you will need for the analysis. The source must extract and assign timestamps and water-

[2] LDBC-SNB Data Generator: https://github.com/ldbc/ldbc_snb_datagen

[3] Pre-generated data: http://www.cs.albany.edu/~sigmod14contest/task.html

marks or epoch numbers to your events. It should also perform any type of cleansing or normalization needed to bring fields into suitable format for analysis.

To simulate a realistic stream source, your code must make sure to serve events proportional to their actual application timestamps. If the events in the generated datasets are too sparse, define a *speedup factor* parameter that adjusts serving speed. For example, a factor of 60 means that events of 60 seconds are served in 1 second.

To make things more interesting, you should additionally have events delayed by *a bounded random amount of time* which causes the events to be served slightly out-of-order of their timestamps. Your queries must ensure result correctness even when events arrive with delay.

Hints:

- The upper bound of event delay will guide how your source generates watermarks and time punctuations.
- You will probably want to use Apache Kafka, a message queue, or some other similar tool of your choice (your local file system might do the job too) for data storage and re-play.

#1. Active posts statistics – 25 points

Write a streaming application that ingests the appropriate streams and continuously computes per-post statistics. For every *active* post, output the following:

- The number of comments, updated every 30 minutes.
- The number of replies, updated every 30 minutes.
- The number of unique people engaged with the post, updated every 1 hour. A person is considered engaged with a post if they have either commented or replied to the post or liked any of its comments.

A post is considered *active* if it has received a comment, reply, or like within the last 12 hours.

Hints:

- Computing this type of continuous statistics can be very memory-consuming if performed naively. For instance, you could think of maintaining active posts as separate 12h-sets containing all post interactions that occur within these time boundaries, i.e. one set for 00:00-12:00, one for 00:30-12:30, another for 01:00-13:00 and so on. Such a strategy can lead to severe data redundancy, especially when the update frequency (30 minutes) is much smaller than the length of the active set (12h). Can you think of ways to reduce data duplication and share results across windows?

#2. Recommendations – 25 points

To keep your users engaged and help them expand their networks, you are launching a new "who-to-follow" recommendation service. The recommendation service analyzes user behavior online and suggests *similar* persons to connect with each other.

For 10 random persons of your choice, compute a list of the top-5 recommended most similar persons. The recommendation algorithm should be based on user activity of the last 4 hours and the list results should be updated every 1 hour.

Make sure to clearly define your similarity metric. Two persons can be considered similar because they discuss similar topics, they have friends who discuss similar topics, they went to the same school, they are members of the same forum, or a combination of the above. Explain how you compute similarity scores and how you use them to rank candidates according to your metric.

Hints:

- Make sure your service does not recommend users who are already “friends”.
- You might want to filter out inactive user accounts from your recommendation list.

#3. Unusual activity detection – 10 points

No online social network likes spam accounts, bots, and fraudulent users. Your last task is to build an application that detects persons with *unusual* activity in the social network and proposes them for account suspension to moderators.

To implement this task you will first need to define what is considered *usual* or *normal* activity and detect posts, comments, or other user actions that do not comply with this definition. To achieve that, you should define a set of *features* that together provide one or more *signatures* of normal activity. These signatures should be continuously updated so that, at any point in time, they reflect the history of the whole stream seen so far.

Hints:

- To detect fake posts, you might want to consider text features, such as the ratio of unique words to the total number of words in the post. You can continuously compute this feature for every post published in the social network and flag posts with a lower-than-average ratio as possibly automatically generated. Note that the *normal* value for this ratio might depend on the post length, so that you will need to maintain a set of features instead of a single one.
- This is indeed a case of anomaly detection and you could identify outliers with a clustering algorithm, such as k-means. How would you apply a clustering algorithm in an online setting? Would you have to recompute its results every time a new event arrives at the system?

Submission deadline: 26 May 2019

Written Reports (30 points)

1. Design document (7.5 points)

Briefly describe *what* you are planning to build and *how* you are planning to build it. Briefly argue why your design solves the given problems and describe its advantages over alternative approaches.

Your design document should answer to the following:

- Where does your application read input from? How does this choice affect latency and semantics (order)?

- What is the format of the input streams for each of the analytics tasks?

- Where does your application output results? How are these results shown to the user?
- What is your stream processor of choice? What features guided your choice?
- Will you use other auxiliary systems / tools, e.g. for intermediate data?

Please submit **one report per team**, maximum **2 pages** .

Submission deadline: 18 March 2019

2. Midterm progress report (7.5 points)

Your midterm report should briefly describe the following:

- Your progress so far. Which tasks have you implemented?
- Any divergence from your original plan. Did you change your mind about a tool you wanted to use or an algorithm?
- Challenges and issues faced so far and how you solved them or planning to solve them.
- Outstanding tasks and a timeline for completion.

Please submit **one report per team**, maximum **2 pages** .

Submission deadline: 29 April 2019

3. Final report (15 points)

Your final report needs to be individual and briefly describe the following:

- A high-level architecture and functionality of your application.
- Supportive documentation including usage instructions. How can I use what you built? What result shall I expect? How can I run tests to verify correctness?
- Your individual contribution to the project. How did you and your partner split the work?
- Your reflection on what you would do differently if you could build this all over again.
- Anything you found fun, interesting, surprising, or annoying while working on this project.

Please submit **one report per person**, maximum **4 pages** .

Submission deadline: 1 June 2019