Tweetoscope

Project overview

The goal of this project is to create a complete data processing pipeline able to predict the popularity of a tweet given the cascade of retweets in a fixed observation window. This application is structured around microservices communicating with each other through Kafka messages

Workflow presentation

We simulate the arrival of tweets with a tweet generator. Those tweets will be collected and grouped into cascades of retweets, those cascades will be sent to a predictor that wants to predict the final number of retweets. Once they are completed, they will be added to a training set used to train the predictors.

The predictions are made using a Hawkes Process and several random forests. The random forest predicts a parameter that gives us a more precise result on the total retweets prediction.

We also have a Dashboard created to look in real time at the tweets most likely to create big cascades. It allows us to predict Twitter trends. Next is the Monitor, when cascades are finished, we can compute statistics about the precision of the previous predictions. It allows us to check in real time that the model still works correctly.

Finally, a Logger receives messages from several critical processing steps of the pipeline. Several categories of logs can be sent, from a simple information to a critical failure message.

Implemented features

Our gitlab pipeline builds and runs on a base image that is different for each branch. Each time there is a change on the requirements or the Dockerfile, the image is rebuilt, while caching from the image from master. This enables us to develop on multiple branches with different images without fearing a mistake on one branch would affect the others. Unfortunately, since the git runner's imagepullpolicy is set to IfNotPresent and since we can't edit it, we sometimes have to rename the base image to force a re-pull.

We used CMake and setuptools to packager our library. CMake automatically reads the python requirements and configures setup.py at build time, ensuring we don't have to manually update the requirements in more than one place.

We also added a upload job to the pipeline to upload our binaries. Unfortunately, since gitlab-student uses gitlab 13.3 and not 13.5, we were unable to upload it directly to gitlab's package repository for releases. However, by pushing a tag to the repository, it uploads the built files to the external repository automatically.

We also made sure to install all the libraries as static libraries, thus saving us the trouble of installing any dependencies on the docker images when deploying the generator and collector. The only exception is rdkafka, which thankfully is light enough.

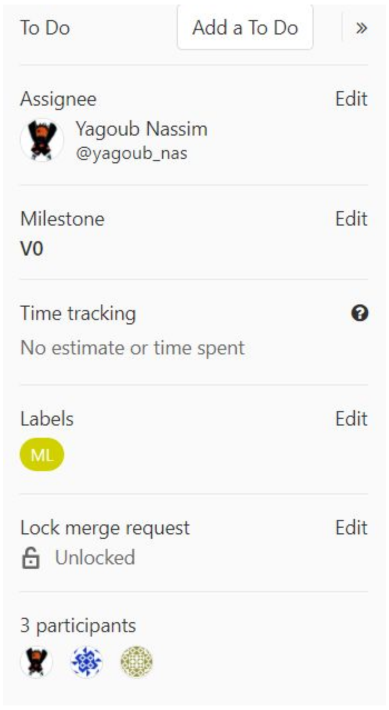
We used Doxygen with Cmake to generate the documentation for both the C++ and Python code, and host it directly on gitlab pages.

We also added logging to all our scripts, using boost log for C++ and Logging for Python, with automatic file rotation, and adding a maximum of information such as filename, linecode, time etc, keeping the same format between Cpp and Python to have an easier time to parse the logs.

For testing, we used Boost test for C++ and unittest for Python, and we run it on an independent stage on the pipeline.

For monitoring, we used Grafana for the dashboards along with Promtail to parse the logs, Loki to query them, and Prometheus for the alerts. We also implemented simple python scripts for the monitor and the dashboard.

Work methodology



To Do — Add a To Do »

Assignee — Edit
Yagoub Nassim
@yagoub_nas

Milestone — Edit
V0

Time tracking
No estimate or time spent

Labels — Edit
ML

Lock merge request — Edit
Unlocked

3 participants

We used gitlab-student.centralesupelec.fr for this project.

We created issues for each important step according to our planning. For example, there were estimator or dashboard issues. We tried to achieve this project with all the features provided by gitlab-student such as the milestones, merge requests or issues. Milestones in gitlab are a way to track issues and merge requests created to achieve a broader goal in a certain period of time, so it was very helpful to organize our issues and merge requests into a cohesive group. We also used the milestones as agile sprints. Indeed, all issues and merge requests were related to a particular sprint. Moreover, merge requests give to the members the possibility to agree and put an “approved” status.

We also used labels to sort and describe issues and pull requests. An issue will receive at least one label and a milestone and a new pull request will receive at least one label.Our labels were ML, Kafka, Devops and Cpp.