FAF.PTR16.1 Spring 2023
Project 1: Stream Processing with Actors

Handed out:	Monday, March 6, 2023
Due:	Friday, April 14, 2023

General Requirements

Compared to the previous Project, all weeks for this one aim to build upon the same application. The goal is to finish the Project with a more or less functional stream processing system.

Since you will be working on a complex application, each presentation will now require you to present 2 diagrams: a Message Flow Diagram and a Supervision Tree Diagram. The Message Flow Diagram describes the message exchange between actors of your system whereas the Supervision Tree Diagram analyzes the monitor structures of your application.

Every task you work on should be easily verifiable. Your system should provide logs about starting / stopping actors, auto-scaling / load balancing workers and printing processed tweets on screen.

P1W1

Minimal Task Initialize a VCS repository for your project.

Minimal Task Write an actor that would read SSE streams. The SSE streams for this lab are available on Docker Hub at `alexburlacu/rtp-server`, courtesy of our beloved FAFer Alex Burlacu.

Minimal Task Create an actor that would print on the screen the tweets it receives from the SSE Reader. You can only print the text of the tweet to save on screen space.

Main Task Create a second Reader actor that will consume the second stream provided by the Docker image. Send the tweets to the same Printer actor.

Main Task Continue your Printer actor. Simulate some load on the actor by sleeping every time a tweet is received. Suggested time of sleep – 5ms to 50ms. Consider using Poisson distribution. Sleep values / distribution parameters need to be parameterizable.

Bonus Task Create an actor that would print out every 5 seconds the most popular hashtag in the last 5 seconds. Consider adding other analytics about the stream.

P1W2

Minimal Task Create a Worker Pool to substitute the Printer actor from previous week. The pool will contain 3 copies of the Printer actor which will be supervised by a Pool Supervisor. Use the one-for-one restart policy.

Minimal Task Create an actor that would mediate the tasks being sent to the Worker Pool. Any tweet that this actor receives will be sent to the Worker Pool in a Round Robin fashion. Direct the Reader actor to sent it's tweets to this actor.

Main Task Continue your Worker actor. Occasionally, the SSE events will contain a "kill message". Change the actor to crash when such a message is received. Of course, this should trigger the supervisor to restart the crashed actor.

Bonus Task Continue your Load Balancer actor. Modify the actor to implement the "Least connected" algorithm for load balancing (or other interesting algorithm). Refer to this article by Tony Allen.

P1W3

Minimal Task Continue your Worker actor. Any bad words that a tweet might contain mustn't be printed. Instead, a set of stars should appear, the number of which corresponds to the bad word's length. Consult the Internet for a list of bad words.

Main Task Create an actor that would manage the number of Worker actors in the Worker Pool. When the actor detects an increase in incoming tasks, it should instruct the Worker Pool to also increase the number of workers. Conversely, if the number of tasks in the stream is low, the actor should dictate to reduce the number of workers (down to a certain limit). The limit, and any other parameters of this actor, should of course be parameterizable.

Bonus Task Enhance your Worker Pool to also implement speculative execution of tasks.

P1W4

Minimal Task Continue your Worker actor. Besides printing out the redacted tweet text, the Worker actor must also calculate two values: the Sentiment Score and the Engagement Ratio of the tweet. To compute the Sentiment Score per tweet you should calculate the mean of emotional scores of each word in the tweet text. A map that links words with their scores is provided as an endpoint in the Docker container. If a word cannot be found in the map, it's emotional score is equal to 0. The Engagement Ratio should be calculated as follows:

$$engagement\_ratio = \frac{\#favourites + \#retweets}{\#followers} \tag{1}$$

Main Task Break up the logic of your current Worker into 3 separate actors: one which redacts the tweet text, another that calculates the Sentiment Score and lastly, one that computes the Engagement Ratio.

Main Task Modify your current implementation of the Worker Pool to make it generic. A Generic Worker Pool should be able to create a pool of Workers specified at initialization. This modification should allow for the creation of an arbitrary number of Worker Pools. Depending on your current implementation, there might be a lot of things to change (or not).

Main Task Create 3 Worker Pools that would process the tweet stream in parallel. Each Pool will have the respective workers from the previous task.

Bonus Task Add functionality to your system such that it would allow for computing the Engagement Ratio per User.

P1W5

Minimal Task Create an actor that would collect the redacted tweets from Workers and would print them in batches. Instead of printing the tweets, the Worker should now send them to the Batcher, which then prints them. The batch size should be parametrizable.

Main Task Create an actor that would collect the redacted tweets, their Sentiment Scores and their Engagement Ratios and would aggregate them together. Instead of sending the tweets to the Batcher, the Worker should now send them to the Aggregator. It should then send the data to the Batcher if a matching set is found.

Note: If your system does not have multiple types of workers, you'll need to go back and add them.

Main Task Continue your Batcher actor. If, in a given time window, the Batcher does not receive enough data to print a batch, it should still print it. Of course, the actor should retain any existing behaviour. The time window should be parametrizable.

Bonus Task Modify your current system to be able to handle retweets. Notice that some tweets are actually retweets and contain a special field `retweet_status`. The system should extract the retweets and treat them as separate tweets.

Bonus Task Modify your current implementation of Aggregator / Batcher communication. Implement the "Reactive Pull" backpressure mechanism between the actors.

P1W6

Minimal Task Create a database that would store the tweets processed by your system.

Minimal Task Continue your Batcher actor. Instead of printing the batches of tweets, the actor should now send them to the database, which will persist them.

Main Task Continue your database implementation. Persist Users and Tweets in separate tables or collections. Make sure not to lose which user posted which tweets.

Bonus Task Continue your Batcher actor. Implement a resumable / pausable transmission towards the database (e.g. in case of DB unavailability).

Good Luck!