Networked Software for Distributed Systems

Politecnico di Milano
Luca Mottola and Alessandro Margara

Project Work 2023/2024
v.0.1

General Rules and Guidelines

1. Projects must be developed in groups of **exactly three students**. Any exception to this rule must be explicitly approved by Luca Mottola.
2. If you are not part of a group of three students and did get approval that you can work in a group of two or alone, **you cannot take the exam**. Again this applies regardless of when in the academic year you plan to work on and/or present the projects.
3. The projects described below are valid for this academic year only. This means that they must be presented and discussed with the course instructors **before the start of the next academic year, that is, roughly the end of September 2024** .
4. Students are expected to demonstrate their projects working in an **actual distributed setting**; for example, using
   - two or more laptops (or virtual machines);
   - cloud deployments;
   - network simulators whenever applicable.
5. Students are expected to present the software architecture and the algorithms/protocols implemented in the projects. They can use a few slides to support their presentation. Preparing slides is, however, **optional**, as long as students are well prepared to present their work.
6. You are to select a combination of projects that, taken together, **covers all 6 technologies** discussed in the course.
7. Each student in a group is expected to answer **questions on any part of any project** developed by the group. This requires knowledge of design principles, implementation choices, and testing procedures of any project developed by the group by any of its members.

8. Discussion of the projects is to be carried out by scheduling an appointment with the relevant instructor. Please drop an email to Luca Mottola <luca.mottola@polimi.it> or Alessandro Margara <alessandro.margara@polimi.it> for this.
9. When asking for an appointment, you also need to **attach the complete source code of all projects and a short design document (~4 pages) for each project**, illustrating the main design and implementation choices.
10. Any question or clarification required on the project description below is to be posted to the public WeBeep forum of the course. We may update this document over time in case further information is needed.
11. The possibility remains for students interested in carrying out their thesis work with either instructor, to skip the project corresponding to their thesis topic. This option must be discussed with the relevant instructor beforehand.

Project #1: Group Monitoring in Mobile IoT (Luca Mottola)

Description

People roaming in a given location carry IoT devices. The devices use the radio as a proximity sensor. Every time two such devices are within the same broadcast domain, that is, at 1-hop distance from each other, the two people wearing the devices are considered to be "in contact". When at least three people are in contact with each other, that is, each person is in contact with the other two, a "group" is formed. Formation of a group must be reported to the back-end as soon as possible. When a further person joins the group, that is, she is in contact with every other current member of the group, a notification carrying the new cardinality of the group is reported to the back-end. Likewise, when a person leaves a group, a notification carrying the updated cardinality of the group is reported to the back-end. The back-end should periodically compute statistics on every group, including i) lifetime, that is, for how long the group existed, and ii) average, maximum, and minimum cardinality so far.

Optional: consider every person to be characterized by her nationality and age.
The back-end should also compute:

1. A 7 days moving average of participants' age per nationality, computed for each day
    - For a given nationality, consider all the groups in which at least one participant is of that nationality
2. Percentage increase (with respect to the day before) of the 7 days moving average, for each day
3. Top nationalities with the highest percentage increase of the seven days moving average, for each day

Assumptions and Guidelines

1. The IoT devices may be assumed to be constantly reachable, possibly across multiple hops, from a single static IoT device that acts as a IPv6 border router, that is, you don't need to consider cases of network partitions.
2. The IoT part may be developed and tested entirely using the COOJA simulator. To simulate mobility, you may simply move around nodes manually.
3. Monitoring of the group membership may be implemented with periodic functionality. The case of a person leaving a group may be handled with a timeout.

Technologies

ContikiNG + Node-RED (+ Spark for the optional part)

Project #2: A Simple Stream Processing System

Description

You are to write a simple stream processing system using actors. Consider a pipeline of three operators. Each message flowing through the pipeline contains a <key, value> pair. Each operator exists in multiple instances. The key space is partitioned across operator instances. Each operator is defined using a window (size, slide) and an aggregation function to process values in the current window. When the window is complete, the operator computes the aggregate value and forwards the result to the next operator. It is possible to change the key when forwarding the value

For example, consider a pipeline to process sensor data. Keys represent the type of data: temperature, humidity, ... Values are the actual sensor reading. Aggregation functions are operators like average, max, min, ... The key determines what operator instance processes what type of data.

The pipeline must be fault-tolerant and ensure that each <key, value> pair is processed *at least once*.

Assumptions and Guidelines

1. Although processes can crash, you can assume that channels are reliable.
2. You are allowed to use any Akka facility, including the clustering/membership service.
3. You need to implement ways to generate faults at any stage of the pipeline, to ensure you can demonstrate the fault-tolerant property.

Technologies

Akka

Project #3: A Model for a Population of Fish

Description

Scientists increasingly use computer simulations to study complex phenomena. In this project, you are to implement a program that simulates the behavior of a population of fish. The program considers N fish that move in a lake (modeled as a 3D cube) with linear motion and velocity v (each fish following a different direction). Each fish has a size. When two fish are close to each other and they are not of the same size, the biggest fish eats the smallest one and its size increases by one.

The program outputs, at the end of each simulated day, the number of fish in the simulation, the size of the smallest fish, and the size of the largest fish.

A performance analysis of the proposed solution is appreciated (but not mandatory). We are interested in studies that evaluate (1) how the execution time changes when increasing the number of fish and/or the size of the lake; (2) how the execution time decreases when adding more processing cores/hosts.

Assumptions and Guidelines

1. The program takes in input the following parameters:
    - N = number of fish
    - L = side of the 3D cube where individuals move (in meters)
    - v = moving speed for an individual
    - d = maximum eating distance (in meters): a fish that is closer than d to at least one bigger fish, gets eaten
    - t = time step (in seconds): the simulation recomputes the position of fish with a temporal granularity of t (simulated) seconds
2. You can make any assumptions on the behavior of fish when they reach the boundaries of the lake (for instance, they can change direction to guarantee that they remain in the area)
3. You can make any assumptions on the behavior of a group of fish that are all within a distance of d (for instance, the biggest fish of the group eats all the others)

Technologies

MPI or Apache Spark

Project #4: Analysis of SMART Data

Description

In this project, you have to implement a program that analyzes open datasets to study the behavior of hard drives. The program starts from the dataset of hard drives test data from Backblaze (data for year 2023) [1] and computes the following queries:

4. 30 days moving average of failures for each vault, computed for each day
5. Percentage increase (with respect to the day before) of the 30 days moving average, for each vault and for each day
6. Top 5 vaults with the highest percentage increase of the seven days moving average, for each day

A performance analysis of the proposed solution is appreciated (but not mandatory). In particular, we are interested in studies that evaluate (1) how the execution time changes when increasing the number of vaults and/or the number of hard drives per vault; (2) how the execution time decreases when adding more processing cores/hosts.

Assumptions and Guidelines

1. You are free to make any assumption regarding missing values.
2. You can assume 0 failures for any day not covered in the dataset, in particular for any day before the beginning of the dataset.

Technologies

Apache Spark or MPI

---

[1] https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data

Project #5: Online services for continuous evaluation

Description

In this project, you are to implement the online services that support university courses adopting continuous evaluation. The application consists of a frontend that accepts requests from users and a backend that processes them. There are three types of users interacting with the service: (1) *students* enroll for courses, submit their solutions for projects, and check the status of their submissions; (2) *admins* can add new courses and remove old courses; (3) *professors* post projects and grade the solutions submitted by students. The backend is decomposed in four services, following the microservices paradigm: (1) the *users* service manages personal data of registered students and professors; (2) the *courses* service manages courses, which consists of several projects; (2) the *projects* service manages submission and grading of individual projects; (3) the *registration* service handles registration of grades for completed courses. A course is completed for a student if the student delivered all projects for that course and the sum of the grades is sufficient.

Assumptions and Guidelines

1. Services do not share state, but only communicate by exchanging messages/events over Kafka topics
    - They adopt an event-driven architecture: you can read chapter 5 of the book "Designing Event-Driven Systems"[1] to get some design principles for your project
2. Services can crash at any time and lose their state
    - You may simply implement state using in-memory data structures
    - You need to implement a fault recovery procedure to resume a valid state of the services
3. You can assume that Kafka topics cannot be lost
4. You can use any technology to implement the frontend (e.g., simple command-line application or basic REST/Web app)

Technologies

Apache Kafka

---

[1] Book available for free at: https://www.confluent.io/designing-event-driven-systems/