Networked Software for Distributed Systems

Politecnico di Milano
Luca Mottola and Alessandro Margara

Project Work 2024/2025
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 mid-September 2025** .
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. Please do so **at least two weeks** before the date you intend to schedule the discussion. If you have any specific constraint, for example, you're an exchange student and must complete the course before a given date, please take this into account. It may be difficult to find a slot that works for everyone.
9. A few days before the discussion, we will ask you to send us **an email attachment with the complete source code and a short design document (~4 pages) for each project**, illustrating the main design and implementation choices. You do not need to send this material already when scheduling the appointment.
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: Understanding RPL Routing

Description

RPL routing is based on building and maintaining a spanning tree on top of the physical wireless topology. You are to implement an IoT application that periodically reports statistics on the RPL routing tree to a back-end. Based on this information, the back-end must be able to reconstruct i) the complete wireless topology, not just including the current routing tree, but also *all neighboring relations*, ii) the routing tree as it is at a given point in time. As the routing tree possibly changes, this information must be accordingly updated at the back-end. The back-end should periodically compute statistics on the current wireless topology and RPL tree, starting when the system booted. These statistics must include i) average, maximum, and minimum depth of the tree, ii) average network diameter, iii) average, maximum, and minimum number of neighbors for any given node.

You need to be able to demonstrate the functioning of the system as described above when operating with either the OF0 or the MRHOF objective function. We expect the documentation to critically discuss the differences in behavior of the RPL protocol, based on the statistics computed at the back-end, depending on the objective function in use.

Assumptions and Guidelines

1. 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 changes in the physical topology, you may simply move around nodes manually.

Technologies

ContikiNG for the IoT simulation, Node-RED or Akka for the back-end.

Project #2: A Simple Stream Processing System

Description

You are to write a stream processing system. 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. Each operator is also provided with an additional input signal that may be used to reset, at any point during the execution, the current state to an empty window.

For example, consider a pipeline of three operators 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. It is possible to change the key when forwarding the value.

Input data should be read from Kafka queues and should be written to Kafka queues. Likewise, operators in the pipeline communicate with each other using intermediate Kafka queues.

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

Assumptions and Guidelines

1. Processes can crash.
2. You are allowed to use any Akka facility, including the clustering/membership service.
3. Kafka queues are replicated and you can assume them to be fault-tolerant.
4. Kafka queues are partitioned, and you should exploit data partitioning to maximize read rate.
5. 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 or Node-RED to implement operator instances, Kafka to implement communication.

Project #3: Heat Diffusion Simulation

Description

Implement a parallel simulation of heat diffusion on a 2D grid using MPI. Heat diffusion, or heat conduction, is the process by which heat energy spreads from high-temperature regions to low-temperature regions within a material. This behavior can be simulated by discretizing the material into a grid and calculating temperature values over time based on the heat equation:

$$\frac{\delta T}{\delta t} = \alpha\left(\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2}\right)$$

where: T(x, y, t) is the temperature at position (x, y) at time t; and α is the thermal diffusivity constant of the material.

Using finite differences, we can approximate the change in temperature at each grid point over discrete time steps. For a given grid point (x, y), the temperature T at the next time step T(x, y, t+1) can be calculated as:

$$T(x, y, t+1) = \alpha\, \Delta t\, T(x, y, t)\left(\frac{T(x+1, y, t) + T(x-1, y, t) + T(x, y+1, t) + T(x, y-1, t)}{\Delta x^2}\right)$$

where: **Δ**t is the time step and **Δ**x is the spatial step size.

For each time step, save the temperature of each point of the 2D grid in a file. At the end of the simulation, use Spark to compute the following queries:

- For each point, compute the minimum, maximum, and average temperature difference over time
- For each point, compute the difference in temperature over a time window of size 100**Δ**t and slide 10**Δ**t
- For each point, compute the maximum time difference across all windows in the second query

Assumptions and Guidelines

1. The program takes in input the following parameters:
    - α = thermal diffusivity constant (e.g., 0.01)
    - Δt = time step (e.g., 0.001 s)
    - **Δ**x = spatial step (e.g., 0.01 m)
    - L = side of the area of the simulation (e.g., 1m)
2. You can make any assumptions on the initial values of temperature in the area (e.g., 100 degrees in the corners and 0 degrees everywhere else)
3. The simulation terminates either when the change in temperature in each and every point is below a given threshold (convergence) or after a maximum number of time steps.

Technologies

MPI and/or Akka for the simulation, Spark to compute queries; you can indeed implement the same simulation processing in MPI and in Akka, essentially achieving two different implementations for the same functionality, if you decide not to use Akka for project #2.