DPS Project 2025

Distributed and Pervasive Systems Lab Course

May 2025

1 Project Description

This project consists in the design and implementation of DESM (Distributed Energy Supply Management), a simplified simulation of an energy supply network overseen by a renewable energy provider. Figure 1 shows the overall architecture of DESM.

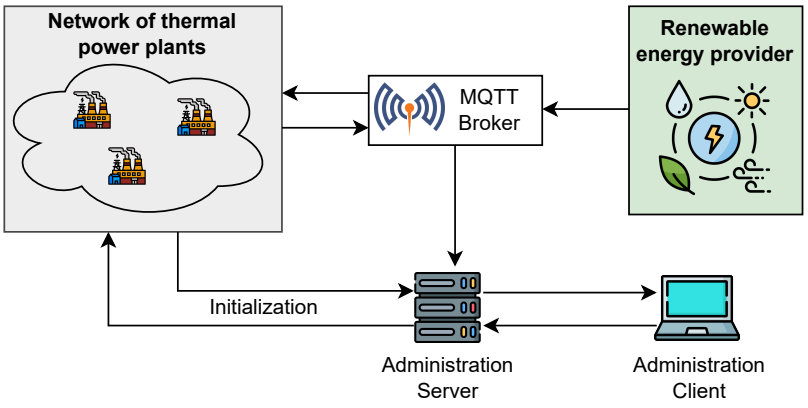

Figure 1: Overall architecture of DESM.

The renewable energy provider generates energy from renewable sources such as wind, hydroelectric, and solar power. Unfortunately, the energy produced is not always sufficient to meet the total demand. To address this shortfall, the renewable energy provider relies on a network of thermal power plants to supply the additional required energy. When a new energy production request is received, the thermal power plants in the network compete to fulfill it. The assignment of the request is based on the lowest offered price among the competing plants. The network of thermal power plants is managed by an administration server, which manages the registration of new plants joining the system. Furthermore, each thermal plant is equipped with sensors that monitor the emission

of pollutants during energy production; these environmental data are periodically transmitted to the administration server, which can be queried by the administration client to retrieve statistics. The goal of this project is to implement the renewable energy provider, a peer-to-peer network of thermal power plants, the administration server, and the administration client.

2 General operation

This section describes the general operation of the system. The later sections will provide a detailed explanation for each specific component.

2.1 Energy request publication

At regular intervals of 10 seconds, the renewable energy provider publishes a request for a certain amount of power, measured in kilowatt-hours (kWh). This request is broadcasted to all thermal power plants within the network.

2.2 Bidding phase

Upon receiving the energy request, each thermal power plant initiates an election process to determine which plant will fulfill the request. During this phase, every thermal plant generates a bid that represents the cost to fulfill the request. The plant offering the lowest cost is selected.

2.3 Request fulfillment

Once a thermal power plant wins the bid, it proceeds to fulfill the energy request. The plant uses an amount of time equal to 1 millisecond multiplied by the number of kWh in the request to complete the energy production. During this period, the plant cannot accept new energy production requests.

2.4 New power plant

The system allows for the dynamic addition of new thermal power plants to the network at any time. Therefore, all algorithms and processes must account for this possibility. For instance, a new thermal power plant may join the network during an ongoing election, and it must be able to participate in the bidding phase.

2.5 Pollution sensors

Once a thermal power plant joins the network, it begins to acquire data from its pollution sensor, which monitors emissions during energy production. These readings are transmitted periodically to the administration server. The administration server can be queried by an administration client so that a potential

competent authority can monitor pollution levels and mitigate environmental impact.

3 Applications to be implemented

For this project, you are required to develop the following applications:

- Renewable Energy Provider: the process that simulates the renewable energy provider.
- Thermal Power Plant: the process that represents the thermal power plants in the DESM network.
- Administration server: a REST server that dynamically adds/removes power plants to DESM and allows the administration client to see the currently active plants in the network and to compute statistics about pollution levels.
- Administration client: a client that allows querying the administration server to obtain information about the currently active thermal power plants in the network and their emissions.

Please note that each power plant is a stand-alone process, and so, it must not be implemented as a thread. In the following, we provide more details about the applications that must be developed.

4 Renewable energy provider

The renewable energy provider generates an energy request every 10 seconds, specifying the required amount of power in kilowatt-hours (kWh) along with a timestamp. The amount of power is a randomly generated number between 5000 and 15000. This request is then published to an MQTT topic, which is subscribed to by all thermal power plants in the network. The request remains on the topic until one plant is selected to provide the requested energy.

5 Thermal power plant

Each thermal power plant runs a process that is responsible for:

- Coordinating with the other plants by using gRPC to decide which plant is going to satisfy the energy request.
- Sending the server information about the values detected by the pollution sensors via MQTT.

5.1 Initialization

A thermal power plant is initialized by specifying:

- ID.
- Listening address and port for communication with the other plants.
- Administration server's address and port.

Note that for the sake of this project, the address of all the components is always going to be *localhost*, as everything is running locally on your machine. Once launched, the plant process must register itself with the system through the administration server. If its insertion is successful (i.e., there are no other plants with the same ID), the plant receives from the administration server the list of the other plants already present in the network (i.e., address and port number of each plant).

Once the power plant receives this information, it starts acquiring data from the pollution sensor. If there are already other plants in the network, the new plant presents itself to the other plants. Finally, the plant subscribes to the MQTT topic in which the renewable energy provider will publish the energy production requests.

5.2 Distributed synchronization

5.2.1 Energy requests managements

The power plants must use a distributed and decentralized algorithm to decide who will take charge of each energy request. Specifically, each request will be handled by the plant that meets the following criteria:

- The power plant must not already be managing another request.
- The power plant is the one offering the lowest price.
- In case two or more plants offer the same price, the plant with the highest ID wins.

5.2.2 Price generation

Each time a thermal power plant has to participate in an election, it uses a random generator to simulate the computation of the price in $\$/kWh$ in the range [0.1, 0.9].

5.2.3 Election algorithm

Since the thermal power plants do not know in advance the price offered by the other plants, you must implement a ring algorithm (Chang and Roberts).

5.3 Pollution Sensor

The power plants are equipped with a pollution sensor that detects the amount of CO_2 emitted during the production process. It is measured in grams (g). During the system's operation, the power plants collect emissions information to be sent to the administration server. Each pollution sensor periodically produces measurements of CO_2 emissions of the plant. Every single measurement is characterized by:

- CO_2 value (g).
- Timestamp of the measurement, expressed in milliseconds.

The generation of such measurements is produced by a simulator. In order to simplify the project implementation, it is possible to download the code of the simulator directly from the page of the course on MyAriel, under the "Project" section. Each simulator assigns the number of seconds after midnight as the timestamp associated with a measurement. The code of the simulator must be added as a package to the project, and it must not be modified. During the initialization step, each plant launches the simulator thread that will generate the measurements for the pollution sensor. Each simulator is a thread that consists of an infinite loop that periodically generates (with a pre-defined frequency) the simulated measurements. Such measurements are added to a proper data structure. We only provide the interface (`Buffer`) of this data structure that exposes two methods:

- `void add(Measurement m)`.
- `List <Measurement> readAllAndClean()`.

Thus, it is necessary to create a class that implements this interface. Note that each power plant is equipped with a single sensor. The simulation thread uses the method `addMeasurement` to fill the data structure. Instead, the method `readAllAndClean`, must be used to obtain the measurements stored in the data structure. At the end of a read operation, `readAllAndClean` makes room for new measurements in the buffer. Specifically, you must process sensor data through the sliding window technique that was introduced in the theory lessons. You must consider a buffer of 8 measurements, with an overlap factor of 50 %. When the dimension of the buffer is equal to 8 measurements, you must compute the average of these 8 measurements. The plant will then send these averages to the administration server.

5.4 Send data to the administration server

Every 10 seconds, each thermal power plant has to communicate to the administrator server the list of the averages of the CO_2 measurements computed after the last communication with the server. This list of averages must be sent to the server associated with:

- The ID of the power plant.
- The timestamp in which the list was computed.

This communication takes place through a proper MQTT topic to which the server is subscribed.

5.5 leaving the network

For the sake of simplicity, it is assumed that no plant, once entered the network, leaves it.

6 Administration server

The administration server collects the IDs of the plants registered to the system and also receives from them the pollution sensor values. This information will then be queried by the administrator client. Thus, this server has to provide different REST interfaces for:

- Managing the initialization of the thermal power plants.
- Enabling the administration client to execute the queries.

In addition, the server must subscribe to the MQTT topic on which the thermal power plants publish pollution sensor values, read them, and store them properly.

6.1 Thermal power plant REST interface

6.1.1 Power plants initialization

The server has to store the following information for each plant joining the network:

- ID.
- Address (in your case it will be localhost).
- The Port Number on which it is available to handle the communication with the other plant processes.

A thermal power plant can be added to the network only if there are no other plants with the same identifier. If the insertion succeeds, the administrator server returns to the power plant the list of plants that have already joined the network, including for each of them the ID, the address, and the port number for communication.

6.2 Pollution Statistics

The administrator server must be able to receive the local CO_2 statistics from the thermal power plants, meaning that it has to subscribe to a proper MQTT topic. These data have to be stored in proper data structures that will be used to perform subsequent analysis. During the development of the project, make sure that you correctly synchronize read and write operations made on these data structures. Indeed, the administration server can receive local statistics while the administrator client is requesting it to perform some computations on such values.

6.3 Administration client REST interface

When requested by the administration client through the interface described in Section 7, the administrator server must be able to compute the following statistics:

- The list of thermal power plants currently in the network.
- The average of the CO_2 emission levels sent by all the plants to the server that occurred between timestamp $t1$ and timestamp $t2$.

7 Administration client

The administration client consists of a simple command-line interface that enables interaction with the REST interface provided by the administration server. Hence, this application prints a straightforward menu to select one of the services offered by the administration server described in Section 6.3 and to enter possible required parameters.

8 Simplifications and Restrictions

It is important to recall that the scope of this project is to prove the ability to design and build distributed and pervasive applications. Therefore, all the aspects that are not strictly related to the communication protocol, concurrency, and sensory data management are secondary. Moreover, it is possible to assume that no nodes behave maliciously. On the contrary, you should handle possible errors in data entered by the user. Furthermore, the code must be robust: all possible exceptions must be handled correctly. Although the Java libraries provide multiple classes for handling concurrency situations, for educational purposes, it is mandatory to use only the methods and classes explained during the laboratory course. Therefore, any necessary synchronization data structures (such as locks, semaphores, or shared buffers) should be implemented from scratch and will be discussed during the project presentation. Considering the communication between the plants' processes, it is necessary to use the gRPC

framework. If broadcast communications are required, these must be carried out in parallel and not sequentially.

9 MQTT Broker

The MQTT Broker on which DESM relies is Mosquitto, and is assumed to be online at the following address: `tcp://localhost:1883`. The renewable energy provider uses this Broker to publish the energy request. The Broker is also used by the thermal power plant to publish the measurments of the pollution sensors.

10 Project Presentation

You must develop the project individually. During the evaluation of the project, we will ask you to discuss some parts of the source code and we will also check if it runs correctly considering some tests. Moreover, we will ask you one or more theoretical questions about the theoretical content of the lab lessons. You will run your code on your machine, while the source code will be discussed on the examiner's machine. You must submit the source code before discussing the project. For the submission, you should archive your code in a .zip file, renamed with your university code (i.e., the "matricola"). For instance, if your university code is 012345, the file should be named 012345.zip. Then, the zip file should be uploaded at http://upload.di.unimi.it. It will be possible to submit your project a week before the exam date. You must complete the submission (strictly!) two days before the exam date at 11:59 PM (e.g., if the exam is on the morning of the 15th, you must complete the delivery before 11:59 PM of the 13th). We suggest the students to use, during their project discussion, the Thread.sleep() instructions to show how the synchronization problems have been correctly handled. We recommend analyzing all the synchronization issues and creating schemes to illustrate how the components of your project communicate. These schemes may represent the format of the messages and the sequence of the communications that occur among the components during the different operations involved in the project. Such schemes can be very helpful during the project discussion. It is not necessary to show the structure of the classes of your project, but only the main aspects regarding the synchronization and the communication protocols you have implemented. During the presentation of the project, we will ask you to test your code by launching at least 4 or 5 plants, the renewable energy provider, the administration server, and a single administration client.

11 Plagiarism

The reuse of code written by other students will not be tolerated. In such a case, we will apply the sanctions described on the course website, in the section

Student Plagiarism of General Information. The code developed for the project must not be published (e.g., GitHub) at least until March of the next year.

12 Optional Parts

In order to encourage the presentation of the project in the first exam sessions, the development of the first optional part becomes mandatory from the September session (included), while both the first and the second optional parts are mandatory from the January session (included).

12.1 First Optional Part: Leaving the network

For this optional part, you must also consider the possibility of a thermal power plant leaving the network. You can assume that each plant process terminates only in a controlled way, meaning that the plant explicitly exits the network. A plant can leave the network in any of the phases described in Section2, by entering the "exit" command into the command line of the process. When leaving the network, the plant process must follow these steps:

- Notify the other thermal power plants.
- Request the administrator server to leave the network.

Note that when a plant receives communication of another plant's exit, it must handle it accordingly.

12.2 Second Optional Part: uncontrolled exit

For the second optional part, you can no longer assume that each thermal power plant process terminates only in a controlled way. Hence, all the distributed synchronization mechanisms implemented for the project must take into account that a plant could leave the system in an uncontrolled way (i.e. Without the ability to notify the provider, the server, or other plants).

13 Updates

If needed, the current text of the project will be updated. Changes in the text will be highlighted. Please regularly check if new versions of the project have been published on MyAriel.