Middleware Technologies for Distributed Systems

Project #1 - Simulation and Analysis of Noise Level

Additional feature: Data cleaning and enrichment

Samuele Negrini, Samuele Pasini, Giorgio Piazza

1 Introduction

The designed system studies the level of noise in different regions of a country. The noise is studied in different ways according to the region:

- **IoT devices**: in some regions the sensor data is assumed to be available, so it is possible to collect data with *IoT sensors* able to measure the noise level according to the distance to noise sources. To simplify computation in this scenario, we assume that a sensor detects the noise level only from its closest noise source and the noise level detected depends only on the distance between the source and the device.
- **Virtual Sensors**: in some regions sensor data are not available, so the noise level detection is simulated via *virtual sensors*, that read data coming from an existing dataset of noise level readings.
- **Computer Simulation**: other regions rely on simulations based on *population dynamics*. In this scenario the noise level is computed for every squared meter of the region considering a certain number of people and vehicles moving in the region.

Data collected in these ways are sent to the backend, where there are data cleaning and enrichment operations.
Incomplete or invalid data are discarded and each reading is associated with the nearest point of interest. We consider data as invalid if the noise level is lower than 10 dB or greater than 180 dB. Data are also invalid if the region ID or the coordinates are unset or the location of the measurement is outside the boundaries of the region.

2 Model of the problem

Before introducing the architecture and the design choices, it is necessary to analyze different parts of the problem.
In particular, we want to shape a well-defined model of the regions and of the data sent to the backend, specifying also assumptions on them. This is necessary because such models must be compatible with all of the three approaches used to compute noise levels.

2.1 Region Model

The three different approaches to collect data require strong assumptions on the structure of the region. We assume regions to be rectangular, the position of noise sensors, noise sources and points of interests is expressed in relative coordinates related to a specific region as shown in figure 1.
Thanks to the rectangular shape, it is possible to define the width and the length of the region and have well defined boundaries given the points {0,0}, {0, MAX_Y}, {MAX_X, 0}, {MAX_X, MAX_Y}.

In the computer simulation we adopted a grid system so the position coordinates become discrete numbers. In particular every cell represents a squared meter of the region and it has a related noise level measurement. We express the position of noise sources with the indices of the grid cell where they are located as shown in figure 2.

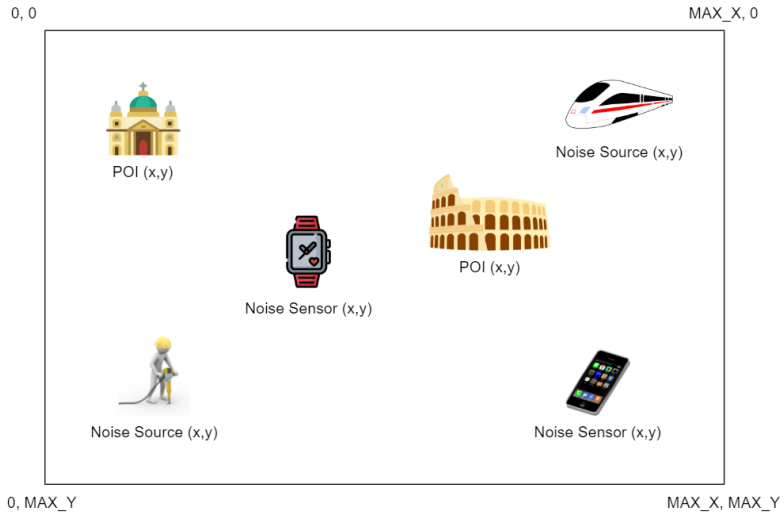

Figure 1: Model of the region

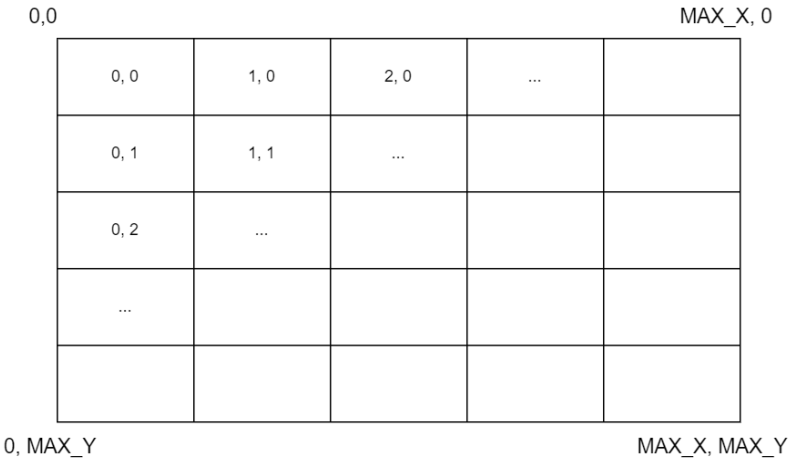

Figure 2: Model of the region for computer simulations

Another strong assumption is related to the dimension of the regions. The computer simulations assumes that every cell is 1 squared meter, so a region with MAX_X and MAX_Y = 1000 will have an extension of 1 squared kilometer.
Due to our limited resources, we assume MAX_X and MAX_Y of every region equals to 100 meters: this is not realistic for a country region but good enough for the simulations in the three different approaches.
We also assume that regions are not contiguous, not crossable and far from other regions, this means that IoT devices will remain in a region, the noise affecting a sensor is inside the same region of the sensor and also the closest point of interest is inside the same region.

2.2 Data Model

According to the requirements, IoT sensors and virtual sensors send noise levels to the backend as the average of the last 6 readings in case it is lower than a threshold **K** (70 dB), or the last 6 raw readings if the average exceeded this threshold. This behaviour is described also in the sent data as an additional parameter in order to specify if the IoT device is sending data as average or raw. If the data is the average, this will be a number expressing the noise level in dB, otherwise the device will send an array of 6 integers.
The devices must send also the X and Y coordinates regarding the position of the last measurement. The coordinates must be positive float values representing the relative position inside the boundaries of the region, otherwise the backend will discard the data.

The region interested by the measurement is specified in the topic of the publication (this will be explained later), and with the triple {x, y, region} every position could be uniquely located in the

country, with the possibility to find the closest point of interest without ambiguity.
This data model could also fit in the computer simulation. The simulation will always send data in average mode, and the coordinates of X and Y will be the indices of the cell affected by the noise detected.

3 Architecture

3.1 Overall architecture

The system architecture is composed of two main systems (Fig. 5):

- **Noise-detection system**: contains all the logic to record noise levels and send them to the backend. The system detects the noise inside the regions, modelled as described before.
- **Backend**: contains all the logic to collect, clean and enrich data.

3.1.1 Noise-detection system

There exists three types of region:

- **Region with IoT devices**: contains real IoT devices like smartwatches and smartphones equipped with noise sensors. These devices will read real noise levels from the surrounding environment. In the region we have also noise sources able to generate noise with a decay related to the distance.
  Since we do not have the equipment for a real scenario, *this region is simulated too.*
  In the simulation, noise sources are wrappers of Cooja disturber motes, able to send radio packets on a specified channel with a specific periodicity. IoT devices detect the radio packets of noise sources and convert the last RSSI to a noise level.

  We are aware that this conversion has no physical meaning but we think this approach could be a good workaround to simulate the real scenario given our resources.

  After the noise reading, the average of the last 6 measures is computed and, depending on the threshold, the average or the 6 raw measurements are sent to the backend using **MQTT**. This requires the usage of a **RPL border router** in the region to connect with the **MQTT broker**.
- **Region with virtual sensors**: we have the same region scenario of the previous one, but in this case the IoT devices does not read radio signal but they *read data* from an existing file representing a sequence of noise detection. Then the data is processed and sent to the backend in the same way as the real IoT devices.
- **Region simulated**: this scenario is completely different from the previous ones: the entire region is completely simulated based on **population dynamics**.

  To obtain the noise level for every squared meter (cell of the region) we need to distribute the noise sources in the grid, calculate their noise production, summing the noise where multiple sources overlap, and move the sources periodically (following a random-movement algorithm).
  This could follow a Map-Reduce approach, with the possibility to distribute the computation.

  In the simulation we have different processes communicating with each-other to implement efficiently the required behavior. At the end of a time-step, every process has one or more cell of the grid for which it is responsible for, and it will send in the MQTT queue the noise level detected in its cells.

  This approach potentially leads to an high number of messages in the queue. The counterpart is to gather all the data in a process reconstructing the final matrix and sending it as a single message.

  The best approach is not totally clear since we do not have the resources to test the simulation in a real scenario with km of extension for the region. We discarded the latter because we think that the computation of the matrix on a single node, with complexity MAX_X*MAX_Y will become the bottleneck of the simulation, and also the message sent will be heavy. In the implemented one, the number of messages is for sure not a problem in case of a sparse matrix (we can discard zero-noise cells), but we will have MAX_X*MAX_Y messages in the

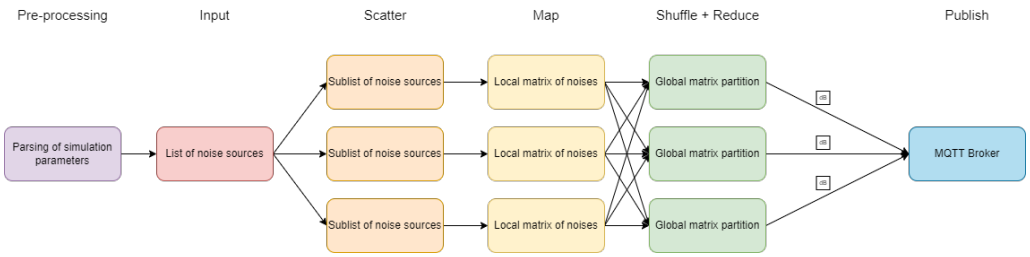

Figure 3: Map-reduce approach implemented

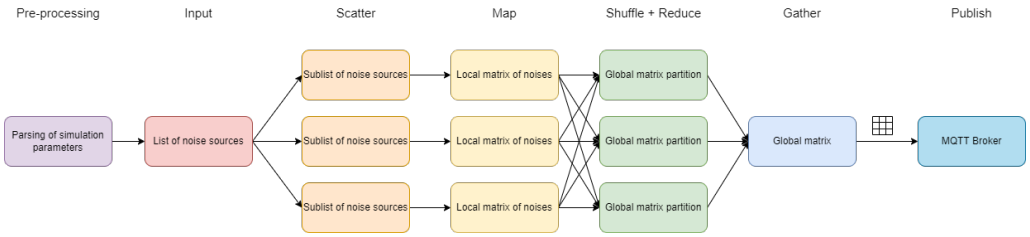

Figure 4: Map-reduce approach with final gather

queue every t seconds in case of a dense matrix. We have also considered that with the implemented behavior the backend is unaware of the region type, while implementing the message with the entire matrix requires to differentiate the behavior for data coming from the simulations.

3.1.2 Backend

The backend subscribes to the *noise/#* topic and listen for incoming message. Upon the receive of a message it casts the message to a JSON object and it performs a first cleaning of messages by validating the object against a JSON Schema. The flows checks if the coordinates of the measurements are inside the boundaries of the respective region and if so proceeds with the computation of the nearest point of interest. Finally it stores the result of the enrichment in the database.

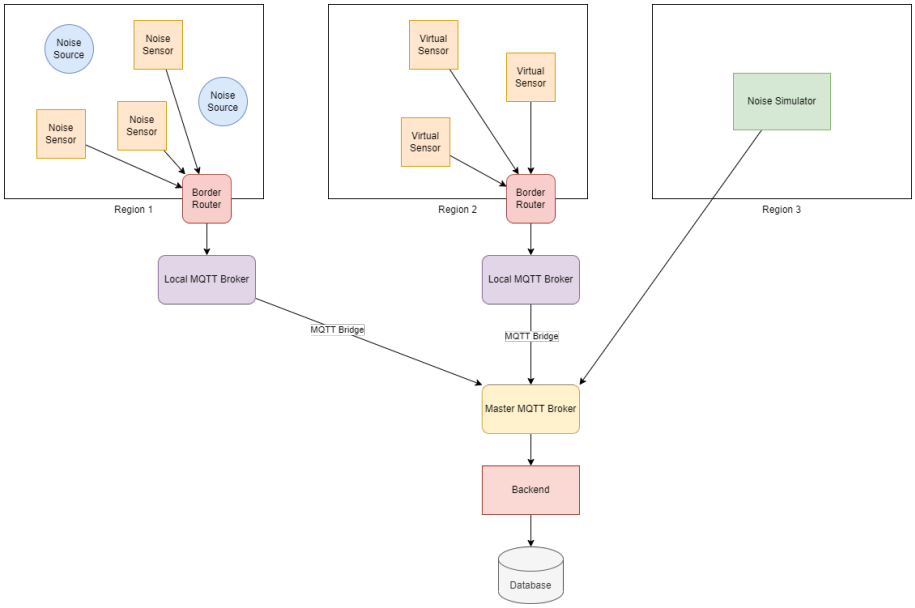

Figure 5: System architecture

3.2 Interfaces

3.2.1 Noise Sensor (Contiki-NG) - Local MQTT broker

Noise sensors inside a region transmit their data to the local MQTT broker. This communication is made possible thanks to the **RPL border routers** that connect the network of IoT devices with the internet.
Each sensor publish messages to the region MQTT topic (e.g. /noise/region1).

3.2.2 Local MQTT broker - Master MQTT broker

Local MQTT brokers bridge all the messages to the master MQTT broker.

3.2.3 Master MQTT broker - NodeRED

NodeRED will subscribe to the master MQTT broker to the topic */noise/#*, where '#' will allow it to receive data from all the regions.

3.2.4 Contiki-NG - File System

Contiki-NG reads the text file in the virtual sensors scenario using a file-system interface named **CFS**.

3.2.5 MPI - MPI

MPI processes are organized into *groups* that exchange messages over a *communicator*. As communicator we used **MPI_COMM_WORLD** that is the predefined communicator that includes all MPI processes. Within a communicator every process has a *rank* and it can use MPI primitives to communicate with other processes. We used *Isend* And *Recv* primitives in point-to-point communication, *Broadcast* primitive and *Scatterv* primitive to split the data.

3.2.6 MPI - Master MQTT broker

Each simulation process publish messages to the region MQTT topic (e.g. /noise/region1) directly to the master MQTT broker.

3.2.7 NodeRED - Database

A module was installed on NodeRED backend to communicate with MySQL database.

4 Design choices

4.1 Contiki-NG

Contiki-NG is an open-source, cross-platform operating system for Next-Generation IoT devices. It can be used in Cooja simulation as the OS of Cooja Motes representing the IoT devices in a region. Contiki-NG focuses on dependable low-power communication and there is the possibility to use standard protocols, such as IPv6/6LoWPAN, RPL.

4.2 Node-RED

We decided to adopt Node-RED as our backend technology. Node-RED is flexible, lightweight and allows us to manage messages received via MQTT in an easy way.
Moreover it allows to import Node.js packages like Ajv JSON which we used to validate the JSON schema. It is also easy to extend Node-RED with a dashboard if it is needed for some visual analysis.

4.3 MPI

MPI is a good choice in compute-intensive tasks like computer simulations. It allows to write a single program executed in parallel on multiple processes that can be distributed.

4.3.1 MPI vs Spark

The Map-Reduce approach described before seems well tackled by Spark, but it has generally a different purpose with respect to MPI. With MPI we have explicit parallelism, communication and synchronization, using low-level primitives and maximizing the usage of resources without fault-tolerance, while in Spark all of that is implicit with high-level primitives easy to use. Given that, Spark is very well suited for *data-intensive tasks*, while we are in a compute-intensive task, where the focus is on the performance.

5 Conclusion

The project aims to show the usage of the technologies explained during the course and their integration in an architecture able to satisfy the requirements, not to have an application ready to be used in a real region of some country. However, the usage of technologies like MPI is not supported by our hardware resources in all of their potential, so the scenarios are extremely poor but the implementation is capable to scale with adequate resources. Regarding IoT devices, we do not have the real devices, and the entire detection behavior is simulated using Cooja Motes and Radio signal, but also in this case the logic is totally fine to work with real noise sensors and sources.

5.1 Future improvements

Despite hardware improvements potentially described before, our implementation could be improved also in other directions. The noise detection could become more realistic, in particular in the Sensor Data scenario. We can be more precise considering all the sources in the range and apply the dB sum formula. Also the sources could be differentiated having different levels of noise at the source. The region model could be improved allowing different shapes and contiguity using an absolute coordinate system (like latitude and longitude). In this way it could be possible to have a nearest point-of-interest from a different region with respect to the one corresponding to the measure. With more resources it is possible to extend the sizes of the scenarios to become more similar to a real country-region environment. Regarding the backend, data analyzed could be visualized in a dashboard to have a better understanding of the noise analysis behavior.