Middleware Technologies For Distributed Systems
Luca De Martini, Luigi Fusco, Arianna Dragoni

Project 1:
Simulation and Analysis of Noise Level

Authors: Luca De Martini, Luigi Fusco, Arianna Dragoni
Date: July-2022
Download page: https://github.com/luigifusco/middleware-noise-level-demartini-dragoni-fusco.git

Contents

1 Introduction

The system analyses the level of noise in a country, by using data gathered by mobile sensors attached to IoT devices. Where sensor data are not available, the system uses data generated by simulations.

To get useful information about the landmarks, the data are analyzed, and then the resulting landmark based data is used to derive large scope trends.

Mobile sensors measure the level of noise periodically in different positions. The readings are averaged over a small time window and sent to the system together with geolocation information.

The data is then used to infer noise estimates about static landmarks, which are the main focus of the subsequent analysis. To derive useful insights about the landmarks, the data are analyzed and correlated to highlight large scope trends over time.

Not every region has the IoT infrastructure to generate useful readings, so in substitution, a simulation can generate estimates of the noise around points of interest in areas where no data is available.

2 Architecture

The general architecture should support a large number of data collection entities, fast cleaning and elaboration of data, and simple querying of results from a unified source. The general architecture should be tiered and easily scalable.

1. Mobile IoT sensors gather mobile readings, and send the collected data to the Edge tier.
2. Edge tier does some processing and aggregation.
3. A backend processes data and computes trends, simulating areas where no data is available.

2.1 Main choices

2.1.1 Split processing

The processing of data is distributed throughout the pipeline. In particular, the IoT devices compute moving average of the readings. The edge cleans data and aggregates inferring information about static locations. It can make use of the locality of data, by partitioning the geographical regions among multiple local edge instances. This makes aggregation easier and response times faster. The edge filters the data before sending it to the backend. This way the processing logic can assume to receive only valid data. The backend does the last processing step: it computes the most important metrics that involve large time windows and multiple locations. Alternatively, a more traditional two tiered architecture could have been used, with data being sent directly to a backend for processing. This however would be less efficient and scalable, because the locality of the data would not be exploited as effectively.

2.1.2 Publisher Subscriber Communication

The communication within the system relies on the publisher subscriber model as a common bus for messages. The IoT and Edge tiers communicate over the lightweight MQTT protocol using Mosquitto servers, while the backend and edge tiers use Kafka as it offers more guarantees and provides additional features. In particular, Kafka is used by the edge to publish enriched data, while the Spark streaming analysis backend pulls and processes it, publishing results to different Kafka topics. This architecture makes it easy to develop other applications that can subscribe to the topics and make use of them, without changes to the rest of the system. The communication is such that microservices can communicate transparently with no knowledge of the architecture. In addition, it can be geographically partitioned, interacts well with Spark, and provides persistency. Instead of using Kafka, an alternative implementation could have used a REST API or RPC to connect edge and backend, this however comes the cost of complexity of development and ease of deployment, compared to an off the shelf publisher subscriber framework.

2.1.3 Streaming data processing

Data processing is performed using Spark's streaming API. A possible approach would have been to save the enriched data in a database, and then make batch computations over it. However, considering the real-time nature of the data, the streaming approach seemed to be a better fit: data can be processed only once while providing lower latency and easy extendability. Also, Spark streaming ensures fast recovery from failures, and if the implementation is extended to more than one worker, it ensures a dynamic load balancing and better resource management, since it balances the workload and launches tasks in parallel.

3 Design

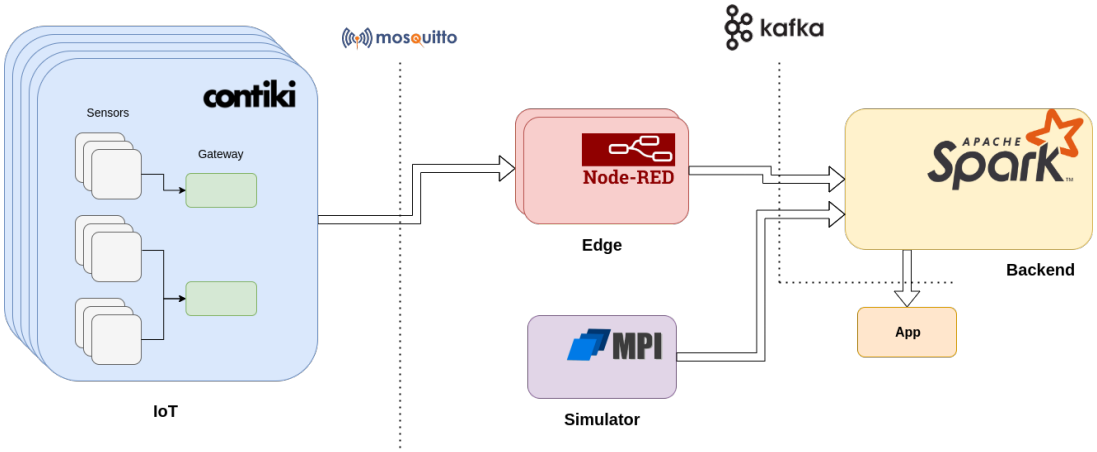

Figure 1: Design of the system, showing how technologies map onto the architecture

3.1 IoT

Mobile devices using ContikiNG act as mobile noise sensors for the system. The data they produce is composed of a windowed average of noise levels, the coordinates of the sensor, and a reliability index. Static gateways using ContikiNG provide internet access to the mobile IoT devices by building an RPL network.

Sensor readings are reported to edge devices via MQTT. A Mosquitto server is present on the gateways acting as a relay. It is configured to forward to the appropriate regional Mosquitto server located on the edge. This way, IoT devices do not know about which edge server they should report to, but are automatically assigned to it based on the gateway they are using.

3.2 Edge

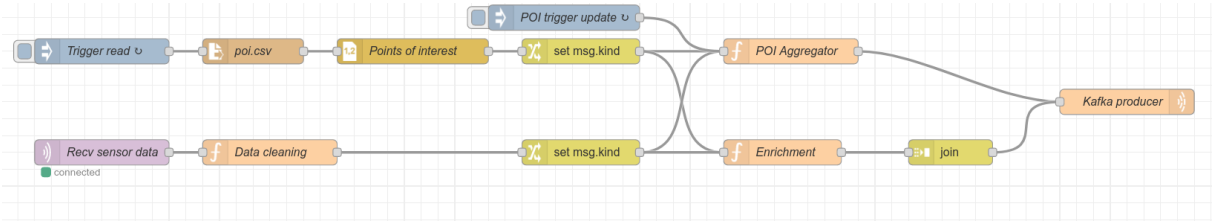

Figure 2: NodeRED flow for data cleaning and enrichment

Edge devices running NodeRED pull sensor data from a regional deployment of MQTT, perform data cleaning, dropping invalid messages and making sure a message contains a noise level, geographical information, and reliability. The data is then enriched with the id of the closest point of interest to the reading location. The edge then performs aggregation using the mobile readings to infer information about static points of interest, computing an estimate of the noise around them.

3.3 Simulation

Noise data for regions without IoT infrastructure is simulated using OpenMPI. The simulation generates data at the point of interest level, publishing directly to Kafka. A region's characteristics are specified through a configuration file. These informations are used to define a 2D space

region in which noise producing entities (vehicles and people) move, each emitting some noise. Due to the physical characteristics of noise, the entities can be partitioned across nodes, each measuring the effect of some entities on all of the points of interest, parallelizing the workload. The overall effect on each point of interest can be computed via reduction, and the results can be published to Kafka by a master node.

3.4 Processing

A Spark cluster running on the backend pulls noise level data, aggregated by point of interests, from Kafka, and then performs large scale analysis on the stream of received measurements. Most of the processing is partitioned by point of interest, working on recent historical data, such as the moving average computations, which perform a reduction on each relevant time window for each point of interest to generate the result. A ranking of the points of interest with the highest levels of noise is computed via reduction on all the point of interest within a time window using a merge sort like strategy: lists are generated and merged though the steps of the reduction, keeping only the highest noise readings, avoiding duplicates for the same point of interest, while bounding the dimension to 10 elements. Finally a stateful map operation keeps track of streaks of good noise levels, publishing to Kafka whenever a new record for a point of interest is generated.