AY 2021/2022

Politecnico di Milano

Middleware Technologies for Distributed Systems

Project report of

# Compute Infrastructure

Students
Matteo Beltrante   Marco Bendinelli   Simone Berasi

Supervisors
Luca Mottola   Alessandro Margara

## 1 Introduction

You are to design and implement a system that accepts compute tasks from clients and executes them on a pool of processes. Each request from a client includes:

- the name of the task to be executed;
- a payload with the parameters to be used in the computation;
- the name of a directory where to store results.

Clients submit compute tasks to a front-end and get notified when they complete. Tasks are scheduled for execution onto a set of processes with this two main characteristics:

- each process can handle one task at a time, so tasks may need to wait until some process is available;
- processes may fail at any time and be restarted, but clients should not be aware of failures, that is, they need to be notified once and only once when the task completes and the results have been successfully stored on the disk.

## 2 Design Choises

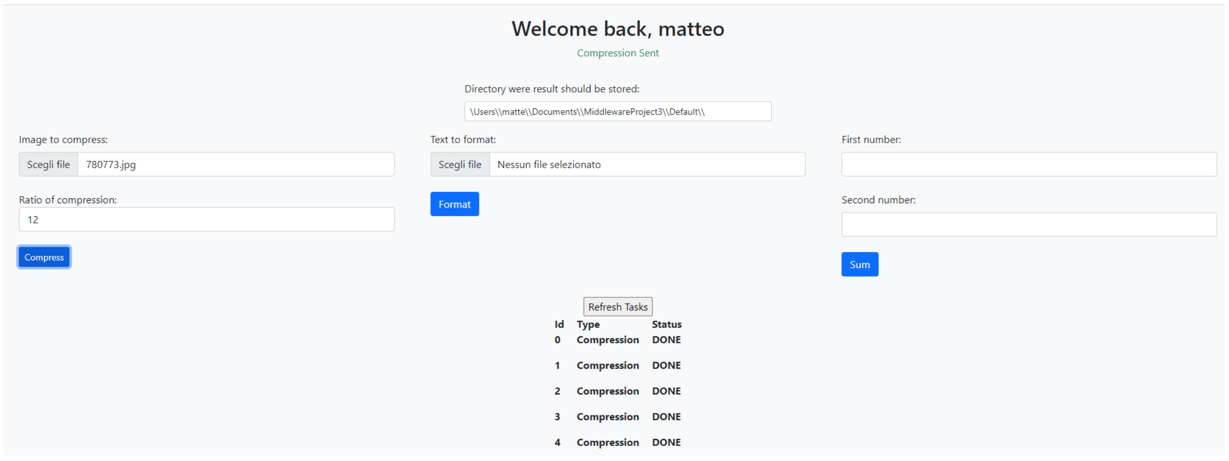
We decided to implement our solution with Akka for the system backend while the client is written as a simple web page using JavaScript and Html.
The backend in particular is composed by a main server actor that handles task processing by rerouting messages onto a pool of working actors, each specialized in a different type of task, this workers do not handle data saving themselves but rely on their actor children to do so.
This approach allowed us to decouple task reception, task handling and data saving thanks to the use of these different actors which are unaffected by each other failures and together create an easily scalable and maintainable system.
At the same time, thanks to the use of the HTTP protocol, the Client is kept unaware of the technology used by the Server and will not be affected by any future changes.

## 3 Architecture

### 3.1 Client

The client is composed of 2 main pages, one for login and one containing 3 forms to collect the data necessary to execute a task and the path to the directory where

to store the result. The second page also contains a list of the user's already submitted tasks with their completion status.
The communication with the server is handled asynchronously with AJAX and JSON while the page elements lifecycle are controlled by a simple JavaScript file.



Welcome back, matteo

Compression Sent

Directory were result should be stored: \Users\\matte\\Documents\\MiddlewareProject3\\Default\\

Image to compress: Scegli file 780773.jpg

Ratio of compression: 12

Text to format: Scegli file Nessun file selezionato

First number:

Second number:

| Id | Type | Status |
|---|---|---|
| 0 | Compression | DONE |
| 1 | Compression | DONE |
| 2 | Compression | DONE |
| 3 | Compression | DONE |
| 4 | Compression | DONE |

Figure 1: Client's form page

## 3.2 Server

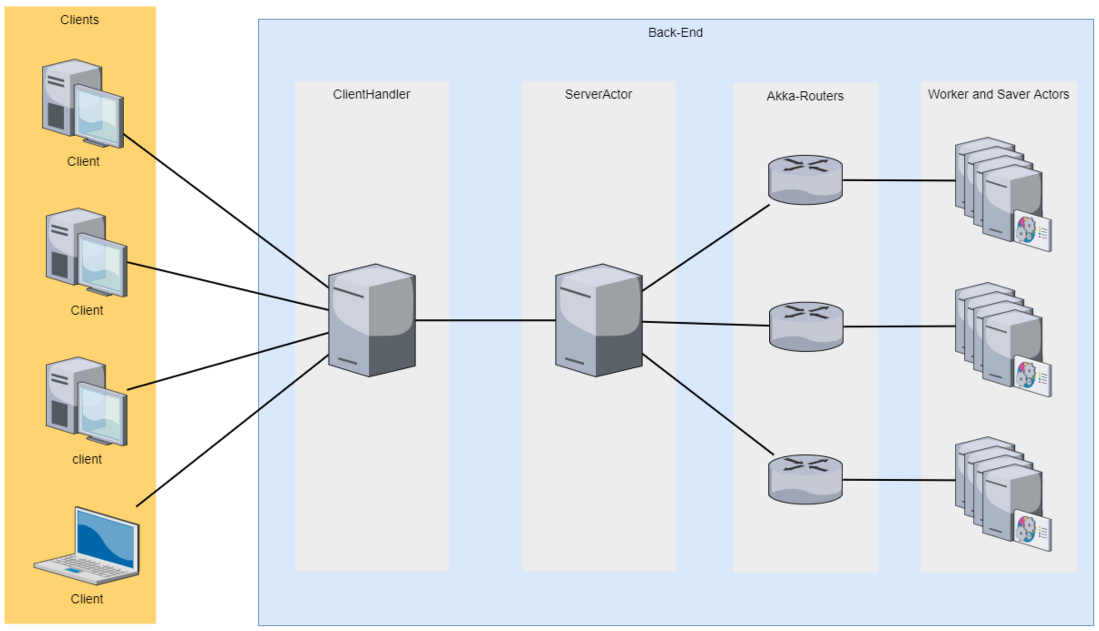


Figure 2: Architecture

As shown in the figure 2 the backend is divided in multiple components:

- **ClientHandler**: it is the exposed interface which has the duty to accept and respond to HTTP requests from the client. When a request to execute a task is received, an appropriate message is asynchronously sent to the main Server Actor; this messages contains all the data necessary to compute the tasks and store the result in the appropriate directory. Using the ask pattern it also query the Server for the user's tasks statuses and forwards them to the client.

- **ServerActor**: this is the main element in our backend, it is implemented as an Akka Actor and is responsible for receiving tasks from the ClientHandler and rerouting them to the appropriate Worker Actors, this is managed with the use of Balancing Akka Routers that allow to redistribute work from busy routes to idle routes. It is also responsible for storing each user task history and current state.

- **WorkerActors**: this actors can be of 3 different types based on the task they are designed to handle and are supervised by their parent Router. Failure are cleanly handled by a custom SupervisorStrategy after resending the current Task to the parent Router in order to avoid its loss. When a Task is finally completed a SaveMessage is created and sent to the SavingActor child.

- **SavingActor**: this actor is in charge of storing the task results in the requested directory, its failures are supervised by his WorkerActor parent in the same way described before. As a failsafe every SaveMessage also store an offset to keep track of how many times it failed and messages over a certain offset are considered permanently failed. When data is finally stored or permanently failed a CompleteMessage is sent to the ServerActor which will update the task status in the simulated DB.

In the figure 3, in the next page, you can see the task life cycle of the program.

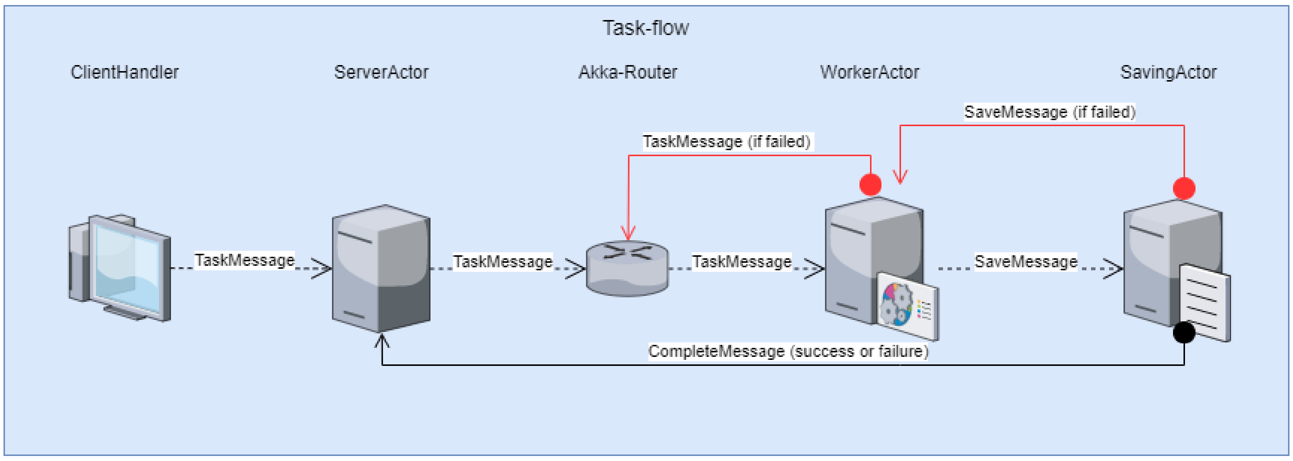


Figure 3: Task flow