universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Sistemas Distribuídos

(ano letivo 2024'25)

Pratical Assigment 2 - Election Day

Today is election day, and `voters` gather at the `polling station` to cast their votes for one of two candidates. `Voters` must wait until the `polling clerk` opens the `polling station` before they can enter. Inside, there is a limited capacity for `voters`; if the station is full, those waiting outside must remain until space is available.

Before voting, each `voter` must present their voting ID, which is validated by the `poll clerk`. If the ID is confirmed as valid and has not been used for voting, the `voter` proceeds to the `e-voting booth` to cast their vote. The vote is randomized but skewed toward one of the candidates. Once voting is complete, the `voter` exits the `polling station`. A `voter` is allowed to vote only once; if the `poll clerk` detects a duplicate voting ID, the `voter` is asked to leave the `polling station`.

At the exit, there is an `exit poll` where some `voters` may be approached by a `pollster` for their opinion. The `pollster` selects a predefined percentage of `voters`—for example, 10%—to inquire about their votes. Responding to the `pollster` is optional, and `voters` are not obligated to disclose the truth about their vote. Their decision is governed by probability. For instance, 60% of approached `voters` may choose to respond, and among them, 20% may provide false information.

Once a `voter` completes their journey through the process (e.g., as illustrated in Figure 1), they may be “reborn” with either a new voting ID or the same ID, depending on probabilistic conditions. A reborn `voter` then re-enters the `polling station` as if they were a new arrival.

Election day terminates when the `polling clerk` announces its end. This can occur after, for example, 500 `voters` have participated or when a set time limit has been reached. The `poll clerk` then closes the `polling station` but allows all `voters` already inside to complete their votes. Once the station is empty, the `poll clerk` informs the `exit poll` that the `polling station` is closed and gathers the votes from the `e-voting booth`.

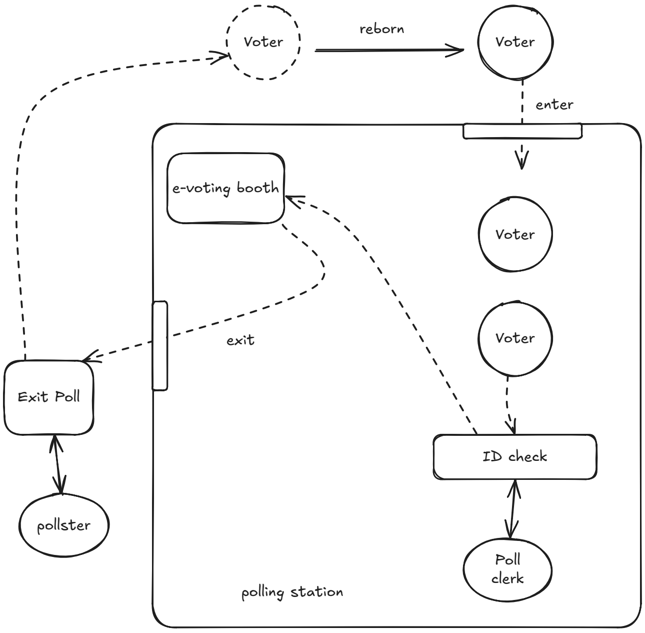

Figure 1: Example workflow of a `voter`.

Objectives and Requirements

Your task is to develop a simulation in Java that models the life cycle of `voters`, `poll clerk`, and the `pollster`. The simulation should utilize TCP sockets and serialization for communication and synchronization between different processes.

Requirements

- The minimum number of `voters` is 3 and the maximum is 10. The number of concurrent `voters` should by pass to the program as an argument.
- The waiting queue, inside the `polling station`, has a minimum size of 2 and a maximum of 5. The queue size should by pass to the program as an argument.
- ID check and voting **do not** follow the entry order of the `voter` into the `polling station`; however, once `voters` arrive at the `ID check` or `e-voting booth`, their actions occur in arrival order.
- The ID validation by the `poll clerk` will take a random amount of time between 5 to 10 milliseconds to execute.
- Casting the vote by the `voter` will take a random amount of time between 0 to 15 milliseconds to execute.
- Responding to the `pollster` will take a random amount of time between 5 to 10 milliseconds to execute.

Additionally, you must implement a log file to track and describe the evolution of the system's internal state. A graphical user interface (GUI) is also required, providing a visual representation of the simulation and reflecting the internal state changes. The GUI can also be used to force the end of the simulation. To be able to visualize several stages of the simulation, you can scale the waiting times to an adequate value.

Guidelines for the implementation

1. Define the structure of the messages exchanged by each representative server in an information-sharing region.
2. Describe the overall organization of the server architecture.
3. Describe the overall organization of the client architecture.
4. Create an interaction diagram that concisely and accurately depicts the dynamics of your solution. Revisit steps 1 through 3 to ensure the diagram is consistent and the description is correct.
5. Implement the solution in Java, using specific reference data types as appropriate.
6. Map the servers and clients onto multiple nodes of the parallel machine. Write the necessary shell scripts to deploy and execute the various application modules.
7. Validate your solution by running multiple tests. For each test, thoroughly inspect the log files to confirm the correctness of the output data.