Politecnico di Milano

090950 – Distributed Systems

Prof. G. Cugola and A. Margara

Projects for the A.Y. 2021-2022

Rules

1. The project is optional and, if correctly developed, contributes by increasing the final score.
2. Projects must be developed in groups composed of a minimum of two and a maximum of three students.
3. The set of projects described below are valid for this academic year only. This means that they have to be presented before the last official exam session of this academic year.
4. Students are expected to demonstrate their projects using their own notebooks (at least two) connected in a LAN (wired or wireless) to show that everything works in a really distributed scenario.
5. To present their work, students are expected to use a few slides describing the software and run-time architecture of their solution.
6. Students interested in doing their thesis in the area of distributed systems should contact Prof. Cugola for research projects that will substitute the course project.

Transactional key-value store

You are to implement a distributed transactional key-value store.

Requirements

Clients submit transactions. Each transaction is a list of read and write operations:

- write(k, v) inserts/updates value v for key k
- read(k) returns the value associated to key k (or null if the key is not present)

The store is internally partitioned and replicated: assuming N nodes, each key (with its values) is replicated in exactly R nodes, with R<=N being a configuration parameter.

Clients can interact with the store by contacting one or more nodes and submitting their transactions, one by one. Nodes internally coordinate to offer the service.
The store should offer sequential replication consistency and serializable transactional isolation. Your design should favour solutions that maximize system performance, in terms of query latency and throughput.

The project can be implemented as a real distributed application (for example, in Java) or it can be simulated using OmNet++. In the first case, you are allowed to use only basic communication facilities (that is, sockets and RMI in the case of Java).

Assumptions

You can assume reliable processes and reliable links (use TCP to approximate reliable links and assume no network partitions happens).

Raft consensus algorithm

Requirements

You are to implement the Raft consensus algorithm, as discussed in this paper: https://raft.github.io/raft.pdf
You can find additional resources and documentation on the algorithm in the official Website: https://raft.github.io/

The project can be implemented as a real distributed application (for example, in Java) or it can be simulated using OmNet++. In the first case, you are allowed to use only basic communication facilities (that is, sockets and RMI in the case of Java).

Assumptions

You can find the assumptions in the Raft paper. In particular, the algorithm is robust against any non-byzantine failure (links can have omissions and delays, processes can stop at any time).

Reliable broadcast library

Requirements

You are to implement a library for reliable broadcast communication among a set of faulty processes, plus a simple application to test it (you are free to choose the application you prefer to highlight the characteristics of the library).
The library must guarantee virtual synchrony, while ordering should be at least fifo.

The project can be implemented as a real distributed application (for example, in Java) or it can be simulated using OmNet++. In the first case, you are allowed to use only basic communication facilities (that is, sockets and RMI in the case of Java).

Assumptions

You can assume a LAN scenario (i.e., link-layer broadcast is available) and you may also assume no processes fail during the time required for previous failures to be recovered.

Virtual beamer in Java

You are to implement in Java a virtual beamer to share a set of slides and show them in a synchronous way under control of a single node (the leader).

Requirements

Users that want to share their presentations create a new session. Other nodes join one of the available sessions (there could be more sessions led by different users). When the session creator decides the presentation to share (choose the format you prefer) the file/files are sent to the joined nodes, then the presentation may start. To send the presentation from the session creator to the other nodes, choose a solution that minimizes network traffic.

Users/nodes may also join while the presentation is running. In this case they choose the node from which to download the presentation in a way to share load.

The slide to show is decided by the session leader (initially the session creator/owner). During a session, the leader may pass its role to another user, which becomes the new leader.

Assumptions

You can assume a LAN scenario (i.e., link-layer broadcast is available). Assume also that nodes (including the actual leader) may crash, in such a case the leader becomes the session creator, or the leader is elected if the session creator also crashed.

In general, privilege a "plug&play" solution that leverages the LAN scenario to avoid the need of entering network addresses and similar information.

Suggestions for the GUI:

- May use the JTextPane class to show html files.
- May use a set of jpg images (one per slide, easy to show in Java)

In general, do not spend much time in implementing the GUI.