Highly Dependable Systems
Sistemas de Elevada Confiabilidade
2020/21

Project - Stage 2
Highly Dependable Location Tracker

The goal of the second stage of the project is to extend the implementation of the first stage to tolerate the possibility that the system may be subject to Byzantine faults affecting server processes. Additionally, the server must provide an additional functionality that allows a user to obtain all the proofs that it provided to other users in a range of epochs.

Besides this additional feature, we will keep the same specification for the Highly Dependable Location Tracker from the previous stage, but enhance the client and server implementations to make them resilient to Byzantine faults that affect both the client and the server replicas.

On the server-side, the single server from the first stage must be replaced with a set of N replicas that collectively implement a Byzantine Fault Tolerant service. The size N of this set must be chosen in such a way that depends on a value f_S, which is a system parameter representing the maximum number of Byzantine faults in the servers, i.e., the maximum number of faults that may occur in servers while preserving the correctness of the HDLT. Additionally, the servers shall protect themselves from Denial of Service attacks, by employing anti-spam mechanisms aimed at reducing the rate at which they may receive computationally expensive requests.

On the client-side, we assume that a malicious user may arbitrarily alter the client library to attack the system, e.g., by sending different reports to different server replicas to break the system guarantees. Students must design and implement solutions to protect the HDLT from such malicious users. It is however outside of the scope of the project to integrate techniques that aim at protecting a legitimate user from a malicious client library (that may, e.g., leak the user’s private key or produce fake replies for the user without even contacting the servers).

Design Requirements

The basic change from the previous design is to replace the client-server communication between the clients and the server with a replication protocol between the clients and a set of server replicas.

All the reports stored for any epoch for a given user must be logically mapped to a (1-N) register, where:

- obtainLocationReport() shall be treated as a read operation with **atomic** semantics
- submitLocationReport() shall be treated as write operations with **atomic** semantics
- obtainUsersAtLocation(pos, ep,...) shall be treated as a set of read operations, each affecting the registers of all the users in the epoch *ep*. Each individual read operation shall have **regular** semantics.
- requestMyProofs(userId, set of epochs, …) **(defined below)** shall be treated as a set of read operations, each affecting the registers of all the users in the set of epochs specified as input. Each individual read operation shall have **regular** semantics.

Additionally, the HDLT service must expose an additional method:

- requestMyProofs(userId, set of epochs, …)
  Specification: returns to userId all the proofs that it generated as a witness for other users in the set of epochs specified as the input parameter. This should only return the information for the user that submitted the request.

Moreover, the system should introduce a mechanism meant to combat Denial of Service attacks. This mechanism is aimed at reducing the number of requests servers may receive from malicious clients, by imposing a computational cost to trigger the execution of ”expensive” requests. This cost should require a computational investment from clients performing such requests, disincentivizing them from flooding the servers with unwanted operations. It is up to the students to identify which functions, among the ones made available by the server, are computationally expensive and, thus, worthy of being protected via this mechanism.

Implementation Steps

To facilitate the design and implementation of the system, students are encouraged to break up the project into a series of steps, and thoroughly test each step before moving to the next one. Having an automated build and testing process (e.g.: JUnit) will help students progress faster.

Here is a suggested sequence of steps:

- **Step 1:** Replicate the server, without implementing any scheme to synchronize the state of the various replicas and assuming clients that do not behave maliciously with respect to the server replicas (e.g., by sending different requests to different replicas). This will involve starting N server replicas instead of one, and replacing the client to server communication with a loop that repeats each communication step N times.
- **Step 2:** As for the next step, students are advised to first implement the (1,N) Byzantine Regular in Section 4.7 of the course book (Introduction to Reliable and Secure Distributed Programming, 2nd Edition)
- **Step 3:** Next, consider a transformation from (1,N) Byzantine Regular register to a (1,N) Byzantine Atomic register.
- **Step 4:** The (1,N) Byzantine Atomic register assumes that only the server can be Byzantine and not the clients. Reason on what is the impact of having Byzantine clients and propose, if needed, solutions to cope with such an issue.
- **Step 5:** Next, extend the implementation to introduce the spam combat mechanism.
- **Step 6:** Implement a set of tests that demonstrates the various dependability features integrated in your system.

Submission

Submission will be done through Fénix. The submission shall include:

- A self-contained zip archive containing:
  - The source code of the project and any additional libraries required for its compilation and execution.
  - A set of demo tests/scripts that demonstrate the mechanisms integrated into the project to tackle security and dependability threats (e.g., detection of attempts to tamper with the data, etc).
  - A mandatory README file explaining how to build the system and run the tests/scripts.
- A concise report of up to 6,000 characters addressing:
  - Explanation and justification of the design, including an explicit analysis of the possible threats and corresponding protection mechanisms.
  - Explanation of the integrity guarantees provided by the system.
  - Explanation of other types of dependability guarantees provided.

The deadline is **21/5/2021 at 19:00**. More instructions on the submission will be posted on the course page.