Highly Dependable Systems

Sistemas de Elevada Confiabilidade
2018-2019

HDS Notary

Stage 2

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 both the notary and users' processes.

More precisely, we will keep the same specification for the HDS Notary system from the previous stage, but enhance the notary and client implementations to make them resilient to Byzantine faults.

On the notary side, students will need to replace the single notary server used in the first stage with a set of N replicas which 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, which is a system parameter representing the maximum number of Byzantine faults, i.e., the maximum number of faults that may occur while preserving the correctness of the HDS Notary system. Additionally, the notary servers shall protect themselves from Denial of Service attacks, by employing anti-spam mechanisms aimed at reducing the rate at which they may receive requests from malicious clients.

On the user side, we assume that a malicious user may arbitrarily alter the client library to attack the system, e.g., in order to try to sell their goods twice or to break the system's consistency. Students must design and implement solutions to protect the HDS Notary system 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).

1. Dependability requirements

The basic change to the existing design in this stage of the project is to replace the client-server communication between the library and the server with a replication protocol between the library (acting as a client of that protocol) and a set of server replicas.

Each tuple 〈goodID, userID must be logically mapped to a (1-N) atomic register, where:

- getStateOfGood() shall be treated as a read operation

- intentionToSell() and transferGood() shall be treated as write operations.

As already mentioned, an additional extension to the existing design is the introduction of a mechanism meant to combat spam. This mechanism is aimed to reduce the number of requests servers may receive from malicious clients, by imposing a computational cost behind each `transferGood` request. This cost would require a computational investment from clients attempting to transfer goods, disincentivizing them from flooding the servers with unwanted operations.

2. Implementation steps

To help in the design and implementation, we suggest that students break this task up 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 you 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 non-byzantine clients. 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.

3. Implementation constraints

Given the physical limitations when running the project on a single machine, and **solely for testing purposes**, the Portuguese Citizen Card used to authenticate the notary may be replaced by alternative mechanisms not based on smart-cards (e.g., digital signatures executed by the host machine using private keys residing on a local file-system).

The project should still be able to function as usual, i.e., with the Portuguese Citizen Card, in case the notaries are executing on different machines.

4. 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. The archive shall also include a set of demo applications/tests that demonstrate the mechanisms integrated in the project to tackle security and dependability threats (e.g., detection of attempts to tamper with the data). A README file explaining how to run the demos/tests is mandatory.
- a concise report of up to 4,000 characters addressing:
    - explanation and justification of the design, including an explicit analysis of the possible threats and corresponding protection mechanisms,
    - explanation of the dependability and security guarantees provided.

The deadline is **May 15 at 17:00**. More instructions on the submission will be posted in the course page.