Confiabilidade de Sistemas Distribuídos – 2018/19

Project 2

Byzantine fault-tolerant system for smart contracts

Introduction

The goal of this project is to create a dependable system that can execute (basic) smart contracts. The system to be developed should extend the project to: (1) maintain additional state and be able to execute basic operations on the state; (2) support recovery of the servers.

Architecture and specification

Besides maintaining information about wallet, which can be maintained in clear text, the system should be able to maintain additional state in cyphered mode. The system should support, at least, the following data types:

- HOMO_ADD: an integer, stored in cyphered mode, that supports an add operation with cyphered data:
- HOMO_OPE_INT: an integer, stored in cyphered mode, that supports the retrieval of database keys, whose values are between two values.

For support "basic" smart contracts, the API of the system should support the following operations:

The API of the system should support the following operations for all data types:

- create(key, initial_value, type)
  - Add the (key, value) pair of type "type" to the database. The following types should be supported "HOMO_ADD", "HOMO_OPE_INT", and "WALLET".
- get(key)
  - Returns the value associated with key "key".
- get(key_prf, lower_value, higher_value)
  - Returns all key with key prefix "key_prf" and values between "lower_value" and "higher_value" (inclusive).
  - Simples alternative:
    - get(lower_value, higher_value)
      - Returns all key with values between "lower_value" and "higher_value" (inclusive).
- set(key, value)
  - Set the value of "key" to "value".
- sum(key, value)
  - Sum "value" to the value of "key".
- conditional_upd(cond_key, cond_val, cond, list[(op,upd_key, upd_val)])
  - Executes the list of updates if the condition holds, where the condition is parameterized by "cond", as follows:
    - 0 -> db[cond_key] = cond_value
    - 1 -> db[cond_key] != cond_value
    - 2 -> db[cond_key] > cond_value
    - 3 -> db[cond_key] >= cond_value
    - 4 -> db[cond_key] < cond_value

        - 5 -> db[cond_key] <= cond_value
    - The list of updates is parametrized by “op”, expressing:
        - 0: set value: db[upd_key] = upd_val
        - 1: add value: db[upd_key] = db[upd_key] + upd_val
    - Simple alternative. Implement the following operations:
        - conditional_set(cond_key, val, upd_key, val2)
            - IF db[cond_key] > val
                - db[upd_key] = val2
        - conditional_add(cond_key, val, upd_key, val2)
            - IF db[cond_key] > val
                - db[upd_key] = db[upd_key] + val2

NOTE: the solution to be developed may include additional parameters for any operation – e.g. to provide the data type of the keys.

The system should be able to execute these operations on all data types, either cyphered or plain text.

For a given operation, whenever possible, the system should execute the operations resorting to the cryptographic algorithms used. In the cases where this is not possible, the system should use a service running in a SCONE container (https://sconedocs.github.io/Java/) to execute the operation in an private context.

NOTE: if you cannot run the Java service in SCONE in your machine, just create the service and run it as a normal Java program.

Server recovery and authenticated boot

The system should include a mechanism for executing server recovery, by stopping and restart a server when necessary. To this end, the system should include, in each machine, a service that can be accessed by the administrator with the following operations:

- launch(url, hash, mainclass, parameters)
    - URL: URL for obtaining the jar containing the software to run
    - Hash: hash of the jar to verify integrity
    - Mainclass: Main class to run
    - Parameters: parameters to be used in the main class
- stop(id)
    - id: id of the software to stop, as returned by the launch.

NOTE: for Smart-BFT to continue working, it is necessary that a quorum of replicas is running. The system should avoid stopping multiple replicas at the same time.

Evaluation

The goal of the evaluation is to compare the performance of the different operations for the different data types.
Thus, the benchmarks to run should, for each data type, start by creating (key,value) pair, followed by a sequence of equal operations.

E.g. create (key,value) pairs of type “HOMO_OPE_INT” followed by add operations; create (key,value) pairs of type “HOMO_OPE_INT” followed by get(low, high) operations; etc.

The experiments should be run, preferably, in a distributed setting with 3 machines connected to a local network, running the following software:

- 2 of the machines should run two replicas of the server each;
- the third machine should run a simple benchmark.

Report

The report should present your work and have the following structure:
Introduction : presents the objectives of the project;
System design : presents the architecture of the systems and the algorithms used;
Evaluation : presents the evaluation (note: do not forget to present the experimental setup);
Conclusions.

Dates

27/May – delivery of the code;
31/May – delivery of the report.

Final notes

The projects basically combines the exercises proposed in labs 7-9. Check the materials given in those labs for additional information.