Course Project: Global Snapshot

Distributed Systems 2015 – 2016

In this project, you will implement a simple distributed bank application capable of transferring money between remote branches. For the sake of routine integrity checks, initiated periodically by the bank personnel, the application should support capturing the global state of the system. The captured global state should confirm that the global invariant (total amount of money in the system) holds.

1 The Distributed Banking System

First, implement a bank application that will act as one branch of a distributed bank. Basically, these bank branches just send money to each other at unpredictable times. You will create many instances of this application on different hosts, to form the distributed banking system. Specifically, the bank module:

- contains a list of all the branches, distinguished by an identifier (just a number);
- starts out with an initial balance of $1,000,000;
- sends random amounts of money (within $1–100) to random banks continuously;
- accepts connections from other banks, that are sending money to the current one.

Safe transfers. A reliable bank application may not lose or create additional money, therefore the money transfer operation needs to be acknowledged and should implement the *at most once* semantics. A simple technique described below will guarantee this safety property if only communication failures may occur. Even if some messages get lost, eventually the transfer is completed.

Assume bank A sends S dollars to bank B. First it withdraws S from the local account but memorizes this amount as *unacknowledged*. Then A makes a transfer request to B, inserting a unique transfer ID (e.g., sequence number) into the message. If an acknowledgement is received from B, the transfer is complete. Otherwise, in case of timeout or an error reported by the communication system, A retransmits the same request again, with the same transfer ID, until an acknowledgement is received. The bank B, upon receiving a request, first checks whether it has already served this request based on the transfer ID. If so, the request is a duplicate and should not be executed again. Otherwise, B performs the requested operation and memorizes the transfer ID (associated with the bank A). In any case, B sends an acknowledgement to A.

For simplicity, banks are not allowed to start new transfers before finishing the previous one, therefore, at most one executed transfer ID needs to be memorized for each sending bank.

2 Distributed snapshot

Periodically (e.g., every second) or on user request one of the banks initiates the global snapshot algorithm of Chandy and Lamport seen in class. In our banking system, a global snapshot will consist of the local state of each bank branch, i.e., its balance, and the amount of money in transit on each of the incoming communication channels. Make sure that reception of a token, capturing the bank state and retransmitting the token are done as an atomic operation. Each instance of the bank application should add a line to its individual log file every time the global snapshot algorithm terminates.

A log record should be a space-separated text line, containing three numbers in the following order: the snapshot unique ID, the recorded balance and the sum of all incoming transfers captured while taking the global snapshot. Additional logging (in arbitrary format, for debugging purposes) should be done to a separate file or to the standard output.

3 Other requirements and assumptions

- It should be possible to run instances of the bank application on separate hosts.
- The number of bank branches and the table of their identifiers and addresses should be configurable at compile time.
- You may assume that at most one snapshot is taken at a time.
- The banks should make a new transfer immediately after finishing the previous one.
- If you use RPC, the transfer acknowledgement is simply the return value of the remote function.

4 Miscellaneous

Technology. You are free to implement this project using either plain TCP sockets or RPC. The use of RPC will be considered more positively. You can in principle use any programming language but, unless you are doing it in Java or C, *ask the instructor first*, as your choice might not be suitable.

Grading. You are responsible to show that your project works. You will be penalized for:

- improper use of synchronization mechanisms. You must not use busy waiting, or "sleep" statements to synchronize your code, and you must properly identify and protect the critical sections of your code;
- presentations where you do not show that you understand the algorithms and basic aspects of the concepts you used in your implementation (sockets / RPC, threads, mutexes, etc.)

Provided code. Sample code that you may use in your program is provided. It contains an example of how to run multiple copies of an RPC server on a single computer and create multi-threaded RPC servers.

People. You are expected to implement this project with exactly one other student.

How/what to present

- You MUST contact through e-mail the instructor (gianpietro.picco@unitn.it) AND the teaching assistant (timofei.istomin@unitn.it), well in advance, i.e. a couple of weeks, before the presentation.
- You can present the project at any time, also outside of exam sessions. In the latter case, the mark will be "frozen" until you can enrol in the next exam session.
- The code must be properly formatted. Follow style guidelines (e.g. this one).
- Provide a short document (2-3 pages) explaining the main architectural choices.
- Both the code and documentation MUST be submitted in electronic format via email at least one day before the meeting. The documentation must be a single self-contained pdf/txt. All code must be sent in a single tarball consisting of a single folder (named after your surnames) containing all your source files. For instance, if your surnames are Rossi and Russo, put your source files in a directory called `RossiRusso`, compress it with "`tar -czvf RossiRusso.tgz RossiRusso`" and submit the resulting `RossiRusso.tgz`. Do not include binaries, nor the documentation in the tarball.
- The project will be evaluated for its technical content (algorithm correctness). *Do not* spend time implementing unrequested features — focus on doing the core functionality, and doing it well.
- The project is demonstrated in front of the instructor and/or assistant.

Plagiarism is not tolerated. Do not hesitate to ask questions if you run into troubles with your implementation. We are here to help.