Dr. Annette Bieniusa
Albert Schimpf, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Exercise 8: Programming Distributed Systems (Summer 2020)

Submission

- You need a team and a Gitlab repository for this exercise sheet.
- In your Git repository, create a branch for this exercise sheet, for example with

  ```
  git checkout -b ex8
  ```

- Create a folder named "ex8" in your repository and add your solutions to this folder.
- Create a merge request in Gitlab and assign Albert Schimpf as assignee. If you do not want to get feedback on your solution, you can merge it by yourself.

1 Total-order broadcast and consensus

In the lecture video you have seen, that Total-order broadcast can be implemented using consensus.

a) Show that it is also possible to do the reverse: Describe an algorithm that implements consensus by using total-order broadcast. Your algorithm should provide a *propose* method to clients and trigger a *decide*-event when a proposed value is decided.

b) Assume we change the total-order broadcast algorithm from the lecture and only use best-effort broadcast instead of reliable broadcast. Show that this would make the algorithm incorrect by giving an execution that violates one of the properties of consensus.

c) Consider the following modification to the total-order broadcast algorithm from the lecture. Instead of two sets `delivered` and `received`, we only use a single set `pending`, which should contain the messages that have been received but not yet delivered.

   Why does this "optimization" not work correctly?

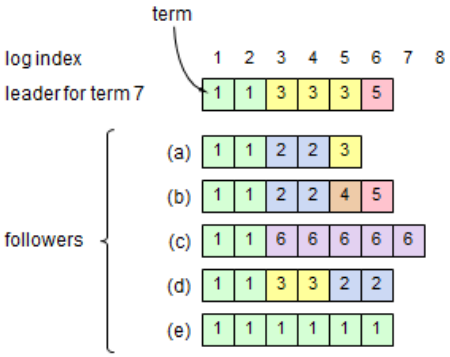
```
State:
  k_p            // consensus number
  pending     // messages received but not a-delivered by process

Upon Init do:
  k_p ← 0;
  pending ← ∅;
Upon to-Broadcast(m) do
  trigger rb-Broadcast(m);
Upon rb-Deliver(m) do
  pending ← pending ∪ {m};

Upon pending ≠ ∅ do
  k_p ← k_p + 1;
  propose(k_p, pending);
  wait until decide(k_p, msg^{k_p})
  ∀ m in msg^{k_p} in deterministic order do trigger to-Deliver(m)
  pending ← pending \ msg^{k_p}
```

2 Raft

These questions are taken from the Raft quizz at `https://ramcloud.stanford.edu/~ongaro/userstudy/quizzes.html`.

To learn about the details of the Raft algorithm, read the Raft paper "In Search of an Understandable Consensus Algorithm (Extended Version)" by Diego Ongaro and John Ousterhout (`https://raft.github.io/raft.pdf`*). You can also watch the video at* `https://youtu.be/YbZ3zDzDnrw`.

a) Consider the figure below, which displays the logs in a cluster of 6 servers just after a new leader has just been elected for term 7 (the contents of log entries are not shown; just their indexes and terms). For each of the followers in the figure, could the given log configuration occur in a properly functioning Raft system? If yes, describe how this could happen; if no, explain why it could not happen.



b) Suppose that a hardware or software error corrupts the nextIndex value stored by the leader for a particular follower. Could this compromise the safety of the system? Explain your answer briefly.

c) Suppose that you implemented Raft and deployed it with all servers in the same datacenter. Now suppose that you were going to deploy the system with each server in a different datacenter, spread over the world. What changes would you need to make, if any, in the wide-area version of Raft compared to the single-datacenter version, and why?

d) Each follower stores 3 pieces of information on its disk: its current term, its most recent vote, and all of the log entries it has accepted.

- Suppose that the follower crashes, and when it restarts, its most recent vote has been lost. Is it safe for the follower to rejoin the cluster (assuming no modifications to the algorithm)? Explain your answer.
- Now suppose that the follower's log is truncated during a crash, losing some of the entries at the end. Is it safe for the follower to rejoin the cluster (assuming no modifications to the algorithm)? Explain your answer.

3 Practical exercise: state machine replication

In this exercise you will implement an online gambling service "Repbet", where users can bet money on the outcome of certain events.

Since gamblers can get pretty angry, when their favorite game is not available, we want to use replication for high availability. We also want high consistency for dealing with money and aggressive gamblers and therefore will use replicated state machines with the Raft algorithm.

The template you can download for this task already contains the basic setup. Check the Readme file for instructions on building the project. The module `repbet_machine` contains the implementation of the replicated state machine and stubs for the functions you need to implement to solve this task. The comments in this file explain how to use and adapt the replicated state machine. We use the library "ra"[1] as Raft library.

[1]Source code is available at `https://github.com/rabbitmq/ra`. You can find some documentation

Implement the following functions:

```
%% Creates a new user with the given Name, Mail, and Password.
%% Returns 'ok' when the user was created and {error, name_taken} if the
%% user name is already taken.
create_user(Node, Name, Mail, Password) -> ...

%% Checks if there is a user with the given name and password combination
%% Returns ok if password matches and {error, authentication_failed} otherwise
check_password(Node, Username, Password) -> ...

%% Creates a new challenge with the given description that users can bet on.
%% Each challenge gets assigned a new unique id, which is returned
%% as {ok, ChallengeId}.
create_challenge(Node, Description) -> ...

%% Makes the given user bet on a given challenge.
%% Result is either true or false (true if the user bets, that the challenge
%% turns out to be true, false otherwise)
%% Money is the amount of money the user bets on this challenge.
%% This amount is directly subtracted from the users balance.
%% Returns ok if betting was successful
%% Returns {error, amount_must_be_positive}, if Money is <= 0
%% Returns {error, invalid_user} or {error, invalid_challenge} if user
%% of challenge does not exist
%% Returns {error, insufficient_funds} if the users account balance is
%% less than the Money he wants to bet
bet(Node, Username, ChallengeId, Result, Money) -> ...

%% Changes the balance of the given user by the given Amount
%% Amount can be positive or negative, but the user account must
%% always have positive value
%% Returns ok on success and {error, insufficient_funds}, if changing
%% the balance would go below 0
change_balance(Node, Username, Amount) -> ...

%% Completes the given challenge by setting a Result
%% Result is either true or false
%% All Money bet on this challenge is distributed among the users
%% that guesses the result correctly,
%% proportionally to the amount they have bet.
complete_challenge(Node, ChallengeId, Result) -> ...

%% Get a list of all challenges, that have not yet been completed
%% Returns a list of tuples: {Id, Description, TotalMoney, YesPercentage}
%% Id is the Id of the challenge
%% Description is its description
%% TotalMoney is the overall amount invested in this challenge
%% YesPercentage is the percentage of money that has been set on 'yes' (true)
list_active_challenges(Node) -> ...

%% Get a list of all challenges, that the given user has participated in
%% Returns a list of tuples:
%% {Id, Description, Result, UserGuess, UserMoney, TotalMoney, YesPercentage}
%% Id is the Id of the challenge
%% Description is its description
%% Result is either 'true' or 'false' for completed challenges
%% or 'not_completed' otherwise
%% TotalMoney is the overall amount invested in this challenge
%% YesPercentage is the percentage of money that has been set on 'yes' (true)
%% UserGuess is the result guessed by the user in his bet
%% UserMoney is the amount of money invested by the user
list_user_challenges(Node, Username) -> ...

%% Get the current balance of a user
%% Returns {ok, Balance} or {error, user_not_found}
get_balance(Node, Username) -> ...
```

at `https://rabbitmq.github.io/ra/`. The interesting parts are in the description of the `ra` and `ra_machine` modules.