ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2013 Prof. R. Wattenhofer

Distributed Systems Part II

Exercise Sheet 4

1 Sequential Consistency: Warm Up

In the following, we consider executions of operations on a single non-replicated memory cell. We assume that there are only two operations:

- $v =$ `read()`: Reads the value currently stored in our memory cell and returns it in v.
- `write(`v`)`: Writes the value v to the memory cell. The write is non-blocking, i.e. it does not wait for the value to be written to the memory, but returns to the client immediately after invocation.

Moreover, we assume that the values written by different invocations of `write()` are all different and unique.

a) Let's assume there is a single client A. Draw an execution E with two operations (both invoked by A) that is *not* sequentially consistent.

b) Add a client B to E that invokes a single operation which makes the resulting execution E' sequentially consistent.

c) Add another operation invoked by client B to E' such that the resulting execution E'' again becomes *not* sequentially consistent. Provide a formal proof that the resulting execution is not sequentially consistent.

2 Sequential Consistency with 3 Clients

We make the same basic assumptions as in the previous exercise: single non-replicated memory cell, the two operations `read()` and `write()` (as above), all writes write unique values.

Definition: Execution E_X is the execution that results after removing client X, together with all the operations invoked by X, from execution E. We thereby allow return values seen by `read()` operations to differ in E and E_X.

Draw an execution E with 3 clients (A, B, and C) that fulfills the following requirements:

- E is *not* sequentially consistent.
- E_A, E_B, and E_C are all sequentially consistent.

3 Three Phase Commit

Three phase commit allows multiple processes to commit or abort a transaction simultaneously. In this task, you have to think about error recovery for 3PC. We assume that processes can crash any time but do not recover and there are no Byzantine processes. The network is asynchronous, messages are neither lost nor reordered. A process crashing during a broadcast may send its message to a subset of receivers. A process broadcasting requests can already receive replies even if not all processes have yet received the request.
Processes are asked to make the final step together, this is called non-blocking property:

NB: if any operational process is uncertain, then no process can decide to commit.

Remember, 3PC runs in 6 steps:

1. The coordinator sends `VOTE` to all participants.
2. Participants answer with `YES` or `NO`.
3. The coordinator sends `ABORT` or `PREPARE`.
4. Participants send `ACK` or abort.
5. The coordinator sends `COMMIT`.
6. Participants commit.

a) For five of these steps, processes have to wait before they can execute them. Write down in which steps which processes have to wait for what messages.

b) If a message does not arrive because its sender has crashed, the waiting process times out. For each of the five steps where processes wait: how should the processes react to a timeout? Make sure NB is not violated. Hint: Can they safely abort or do they have to commit? Must they elect a new coordinator?

c) (optional) Open question for bored people: Assuming crashed processes recover and can remember their last actions. Give a sequence of events where 3PC blocks (meaning: where 3PC gets in a state in which it is impossible to make a decision). Note: if no decision has been made yet, then a newly elected coordinator may choose either to commit or to abort. Once the protocol started it cannot be restarted.