Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

HS 2009 Prof. Dr. Roger Wattenhofer, Thomas Locher, Remo Meier, Benjamin Sigg

Distributed Systems
Theory exercise 3

Assigned: November 20, 2009
Discussion: November 27, 2009

1 Authentication

In the lecture we talked about an algorithm using authentication to reach consensus in an environment with Byzantine processes. See chapter 6, slide 132 ff for a detailed discussion about this algorithm.

a) Modify the algorithm in such a way that in handles arbitrary input. Write your algorithm as pseudo-code. The processes may also agree on a special "sender faulty"-value.

Hint: implement `value` as a set, work with the size of the set.

b) Prove the correctness of your algorithm.

Solution

a) The new algorithm looks like this:

```
if I am P then
  $values \leftarrow \{input\}$
  broadcast "P has input"
else
  $values \leftarrow \{\}$
end if
for $r = 0$ to $f + 1$ do
  for all received values $x$ do
    if $|values| < 2$ and accepted $r$ messages "P has x" with $x \notin values$ then
      $values \leftarrow values \cup \{x\}$
      broadcast "P has x"
    end if
  end for
end for
```

```
if $|values| = 1$ then
  decide item in $values$
else
  decide "sender faulty"
end if
```

b) If P is correct: there is only one message in the system, which is accepted in the first round. There are no other messages, hence for all processes $|values| = 1$.

If P is Byzantine:

- Assume that a correct process `p` adds x to its value set in a round $r < f + 1$: Process `p` has accepted r messages including the message from P. Therefore all other correct processes accept the same r messages plus `p`'s message and add x to their value set as well in round $r + 1$.
- Assume a correct process `p` adds x to its value set in round $f + 1$: In this case, `p` accepted $f + 1$ messages. At least one of those is sent by a correct process, which must have added x to its set in an earlier round. We are again in the previous case, i.e., all correct processes added x to its value set.

2 Randomization

In the lecture we talked about a randomized algorithm reaching consensus in an asynchronous system with Byzantine failures. See chapter 6, slides 137 ff for a detailed discussion about this algorithm.

We assume that there are only crash failures but no Byzantine failures. A crash can happen anytime and broadcasts may not be completed. Crashed processes do not recover.

a) How many crash-failed processes can the algorithm handle?

Hint: Have a close look at the proofs for the validity condition, agreement, and termination.

b) Modify the algorithm to handle more crash failures.

c) How many crash failed processes can your modified algorithm handle?

Solution

a) The algorithm can handle $f < n/8$ failures. To find this result we check the proofs for the validity condition, agreement, and termination for numbers that change:

Validity condition Nothing changes

Agreement Nothing changes

Termination If some process does not set its value randomly, all processes must set the same value, i.e. there must not be $n - 4f$ proposals for 0 and $n - 4f$ proposals for 1. This means that $2 * (n - 4f) > n$, or $f < n/8$.

The reason why this property changes is that Byzantine processes can create two different messages, while simple crashing processes cannot.

b) One solution is to replace "if at least n-4f proposals" by "if at least n-3f proposals". There are other correct solutions which will not be discussed in this master solution.

c) We modify the proof from the lecture. For the agreement property we replace "Every other correct process must have received x at least $n - 4f$ times." by "... at least $n - 3f$" times.

Why? Because $n - 3f$ correct processes had to send their proposal in order for one process to decide. Meaning $n - 3f$ correct processes sent their proposal to any process.

Rewriting the termination property leads to $2 * (n - 3f) > n$, or $f < n/6$.

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, 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) In five of these steps processes have to wait, 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 yet been made, then a newly elected coordinator may choose either to commit or to abort. Once the protocol started it cannot be restarted.

Solution

a) The five steps are:

1. Step 2: Participants wait for VOTE request
2. Step 3: Coordinator waits for votes
3. Step 4: Participants wait for coordinators decision
4. Step 5: Coordinator waits for acknowledgments
5. Step 6: Participants wait for COMMIT

b) The reactions are:

1. no one has yet decided on commit, so abort
2. no one has yet decided on commit, so abort
3. some processes already know the coordinators decision, some do not. Some might already aborted, or have sent ACK. The processes have to elect a new coordinator, the new coordinator has to find out what the decision was and resend the decision to all the processes which do not already know it.
4. a crashed processes did not send an ACK. But the transaction can continue because the correct processes are prepared to commit.
5. the next message must be COMMIT, but committing would violate the non-blocking property as some processes may not yet have received PREPARE. So the processes first have to elect an new coordinator, which has to find out about the decision and inform uninformed processes about it.

c) Assume there are three processes p_1, p_2 and p_3. Process p_1 is the coordinator. All the processes vote `YES` on the transaction. p_1 receives and processes the votes, but p_2 and p_3 detach from p_1 before receiving the `PREPARE` message sent by p_1.

p_2 is elected as new coordinator. It sees both p_2 and p_3 are undecided. Not knowing their state p_2 decides to abort, sending the `ABORT` message. p_3 receives the message, then detaches from p_2. If now p_1 and p_3 become connected they cannot progress: p_1 already decided to commit, p_3 already decided to abort.