ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2016 Prof. R. Wattenhofer

Distributed Systems Part II

Exercise Sheet 1

We categorize questions into four different categories:

Quiz Short questions which we will solve rather interactively at the start of the exercise sessions.

Basic Improve the basic understanding of the lecture material.

Advanced Test your ability to work with the lecture content. This is the typical style of questions which appear in the exam.

Mastery Beyond the essentials, more interesting, but also more challenging. These questions are **optional**, and we do not expect you to solve such exercises during the exam.

Quiz

1 An Asynchronous Riddle

A hangman summons his 100 prisoners, announcing that they may meet to plan a strategy, but will then be put in isolated cells, with no communication. He explains that he has set up a switch room that contains a single switch. Also, the switch is not connected to anything, but a prisoner entering the room may see whether the switch is on or off (because the switch is up or down). Every once in a while the hangman will let one arbitrary prisoner into the switch room. The prisoner may throw the switch (on to off, or vice versa), or leave the switch unchanged. Nobody but the prisoners will ever enter the switch room. The hangman promises to let any prisoner enter the room from time to time, arbitrarily often. That is, eventually, each prisoner has been in the room at least once, twice, a thousand times or any number you want. At any time, any prisoner may declare "We have all visited the switch room at least once". If the claim is correct, all prisoners will be released. If the claim is wrong, the hangman will execute his job (on all the prisoners). Which strategy would you choose...

a) ...if the hangman tells them, that the switch is off at the beginning?

b) ...if they don't know anything about the initial state of the switch?

2 Paxos

You decide to use Paxos for a system with 3 servers (acceptors), which we call N_1, N_2, N_3. There are two clients (proposers) A and B. The implementation of the acceptors is exactly as defined in the script, see Algorithm 1.13. We extended the code of the proposers, such that they now use explicit timeouts. The algorithm is described below, note in particular Lines 2-4 and 12-14.

Algorithm 1 Paxos proposer algorithm with timeouts

```
/* Execute a command on the Paxos servers.
 *
 * N, N': The Paxos servers to contact.
 * c: The command to exexcute.
 * δ: The timeout between multiple attempts.
 * t: The first ticket number to try.
 *
 * Returns: c', the command that was executed on the servers. Note that c' might be
 * another command than c, if another client already successfully executed a command.
 */
suggestValue(Node N, Node N', command c, Timeout δ, TicketNumber t) {
```

Phase 1

1: Ask N, N' for ticket t

Phase 2

2: Wait for δ seconds

3: **if** within these δ seconds, either N or N' has not replied with `ok` **then**
4: **return** **suggestValue**(N, N', c, δ, $t+2$)
5: **else**
6: Pick (T_{store}, C) with largest T_{store}
7: **if** $T_{\text{store}} > 0$ **then**
8: $c = C$
9: **end if**
10: Send `propose`(t, c) to N, N'
11: **end if**

Phase 3

12: Wait for δ seconds

13: **if** within these δ seconds, either N or N' has not replied with `success` **then**
14: **return** **suggestValue**(N, N', c, δ, $t+2$)
15: **else**
16: Send `execute`(c) to every server
17: **return** c
18: **end if**

a) Assume that two users try to execute a command. One user calls **suggestValue**(N_1, N_2, a, 1, 1) on A at time T_0, and a second user calls **suggestValue**(N_2, N_3, b, 2, 2) on B at time $T_0 + 0.5sec$.

Draw a timeline containing all transmitted messages! We assume that processing times on nodes can be neglected (i.e. is zero), and that all messages arrive within less than $0.5sec$.

b) In **a)** we chose artificial initial ticket numbers and timeout values, and we saw that Paxos terminates rather quickly.

Let us look at a more simple selection of these values: Both clients start with the same initial ticket numbers $t_A = t_B$ and timeouts $\delta_A = \delta_B$. Assume that both clients start at T_0. What will happen?

Advanced

3 Improving Paxos

We are not happy with the runtime of the Paxos algorithm of Exercise 2. Hence, we study some approaches which might improve the runtime.

The point in time when clients start sending messages cannot be controlled, since this will be determined by the application that uses Paxos. It might help to use different initial ticket numbers. However, if a client with a very high ticket number crashes early, all other clients need to iterate through all ticket numbers. This problem can easily be fixed: Every time a client sends an `ask`(t) message with $t \leq T_{\max}$, the server can reply with an explicit `nack`($T_{\max}$) in Phase 1, instead of just ignoring the `ask`(t) message.

a) Assume you added the explicit `nack` message. Do different initial ticket numbers solve runtime issues of Paxos, or can you think of a scenario which is still slow?

b) Instead of changing the parameters, we add a waiting time between sending two consecutive `ask` messages. Sketch an idea of how you could improve the expected runtime in a scenario where multiple clients are trying to execute a value!

Extra challenge: Try not to slow down an individual client if it is alone!