Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2017 Prof. R. Wattenhofer

Distributed Systems Part II

Exercise Sheet 3

Quiz

1 Consensus in a Grid

In the lecture you learned how to reach consensus in a fully connected network where every process can communicate directly with every other process. Now consider a network that is organized as a 2-dimensional grid such that every process has up to 4 neighbors. The width of the grid is w, the height is h Width and height are defined in terms of edges: A 2×2 grid contains 9 nodes! The grid is big, meaning that $w + h$ is much smaller than $w \cdot h$. While there are faulty and correct processes in the network, we assume that two correct processes are always connected through at least one path of correct processes. We use the synchronous time model; i.e., in every round every process may send a message to each of its neighbors, and the size of the message is not limited.

Let l be the length of the longest shortest path between any pair of nodes in the grid; i.e., l is the number of edges between those two nodes which are "farthest away" from each other. If there are no failures, l is the distance between two corners, i.e. $l = w + h$.

a) Assume there is no faulty node. Write a protocol to reach consensus! Optimize your protocol according to runtime.

b) How many rounds does your protocol require?

c) Assume that you have an algorithm which solves consensus in time $l + 1$. To save power, up to $w + h$ many nodes are not running – which can be seen as a special type of crash failure. Assume that $w = 7, h = 6$. What is the largest l you can find?

d) Assume there are Byzantine nodes and that you are the adversary who can select which nodes are byzantine. What is the smallest number of byzantine nodes that you need to prevent the system from reaching agreement, and where do you place them?

Advanced

2 Consensus in a Grid: The Algorithm

In **1 c)** we assumed to have an algorithm which works in time $l+1$ even if a certain number of nodes are not running. This algorithm works as follows:

All nodes store from which nodes they already received their initial value. In the first round, every node sends its value to all of its neighbors. In all consecutive rounds, every node only forwards values: If it receives a tuple (u, x) containing the initial value x of a node u, it only forwards the tuple to all neighbors, if this is the first time the node hears the value from u. As soon as there is a round in which a node does not hear a new tuple, the node decides for the minimum of all received values and terminates.

Remark: Recall that we assumed that all correct nodes are connected.

a) Show that this algorithm works correctly; i.e., a node does not terminate before it learned the initial value of all nodes.

b) Show that every node indeed terminates at the latest after $l+1$ many rounds.

c) So far we assumed that we only have a special type of crash failures; i.e., that nodes are "crashed" already at the start of the execution. Assume that you run the algorithm with any type of crash failures; i.e, nodes can crash at any time during the execution. Show that with such failures the algorithm does not always work correctly anymore, by giving an execution and a failure pattern in which some nodes terminate too early!

3 Revisiting Paxos

In the lecture you have seen how Paxos can reach agreement without the need of a single coordinator. Recall that Paxos can handle up to $f < n/2$ many crash failures, where n denotes the number of acceptors (servers). Can Paxos also handle byzantine failures?

Assume that all but one node work correctly. Let this byzantine node u be an acceptor with only the following byzantine behavior: Its memory cell for C (the stored command) is corrupt; i.e., a write to C has no effect. Assume that initially $C = 0$. Otherwise u behaves completely correct.

Assume that there are two clients A, B trying to execute command $C_A = 1$ respectively $C_B = 2$. Show that there is an execution in which A and B will end up with different values, i.e., they fail to agree on the same value.

Mastery

4 Consensus in a Grid... again!

In exercise **1 a)** you had to develop a *deterministic* algorithm which reached consensus if there are no failures. In this exercise we want to show a *tight bound* on the runtime for this problem.

Definition 1 (upper bound). *We call t_U an* upper bound *on the runtime, if we can show that the problem can be solved in time t_U.*

The easiest way to show an upper bound is to design an algorithm which solves the problem in time t_U.

Definition 2 (lower bound). *We call t_L a* lower bound *on the runtime, if we can show, that* no *algorithm exists which solves the problem in less than t_L time.*

This is usually more difficult to show than an upper bound, since it requires an argument why no such algorithm can exist.

Definition 3 (tight bound)**.** *We call a bound t* tight*, if we have an upper bound $t_U = t$, and a lower bound $t_L = t$; i.e., the bounds match. In that case, we know exactly how much time solving a problem requires.*

Your task is to show that $t = (w + h)/2$ is a tight bound on the runtime for consensus if there are no failures! For simplicity, assume that both w and h are even numbers, and that every node knows w and h and its "coordinates" in the grid.

Assume the that one round consists of "send, receive, compute" in this order. I.e., if u sends a message to v in round 1, v receives this message already in round 1.

a) Show an upper bound for the problem by providing an algorithm which runs in $(w + h)/2$ many rounds! (If your solution of **1 a)** terminates in $(w + h)/2$ rounds you're done!)

b) Show a lower bound of $(w + h)/2$.

Hint: Choose some distributions of initial values and show that no algorithm can solve consensus for all these distributions in less than $(w + h)/2$, without violating at least one of the requirements of consensus at least for one distribution.

Hint: We used a similar approach in the proof of Theorem 2.21.