The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 8: Distributed Consensus

Notes

Solutions

36. This is a specific case of the coordinated attack where $n = 2$.

The simple and accepted solution is to show that, given some series of sends and acks(), we can never be certain that the last message sent has arrived. Therefore we can remove that message for the sending process - since it doesn't change anything for that process p_{sender} unless it subsequently receives and acknowledgement and it shouldn't change the result of agreement. This continues until there are no messages. Clearly something isn't right here.

Another way to consider this in simple terms is that since both generals know that their messages may be lost, they will want confirmation of delivery to the other general via a subsequent message. Then, how can the responding general be certain that their acknowledgement has also been acknowledged? They need an acknowledgement of the acknowledgement! .... and hopefully you can see this degrade.

The longer form follows:

We can use a indistinguishably proof here to show that coordinated attack is impossible.

Execution A is indistinguishable form Execution B for some process p if p sees the same things in both executions

If A is indistinguishable from B for p, then b can't tell which of these two possible configurations it is in and returns the same output for both.

Consider some chain of executions $A = A_0, A_1, ..A_k = B$, where each A_i is indistinguishable from A_{i+1} for process p_i.

If we are solving agreement, then it must be the case that since pi outputs the same value in A_i and A_{i+1}, every process must output the same value in A_i and $A_i + 1$.

Induction on k, every process outputs the same value in A and B, even though A and B may be very different executions. Lets use this construction to show that there exists some chain of executions from A, where all inputs are 1 and all messages are delivered and arrive at a configuration where all processes have input 0 and no messages are lost.

let A_0 be the starting configuration above. We build $A_1, A_2, ...A_k$ by pruning messages. Consider A_i and let m be some message that is delivered in the last round in which any message is delivered. Construct A_{i+1} by not delivering m. While A_i and A_{i+1} is distinguishable by the recipient of m, on the assumption that $n > 1$ - there is some other process that can't tell whether m was delivered or not.

Likewise construct A_{k+1} through A_{k+n} by changing the input of a process p_i $0 \mapsto 1$. Lastly, add back in messages A_{k+n} through $A_{k+n+k'}$ in the reverse order in which they were removed.

In this case, if agreement holds it must be the case that all processes agree on the same decision in A_0 and $A_{k+n+k'}$

yet if integrity holds, it must be the case that $A_0 = 0$ and $A_k + n + k' = 1$. So either agreement or validity is violated in some execution.

37.

See excalidraw. The key point here is that we need to be specific in our construction when choosing what the lying (byzantine) process says.

38.

(a) Yes, some process pj broadcasts its message. The other processes then run BA with message they received as input.

(b) Yes, each process runs BB as the initializer. Then after n rounds (one for each process), take the majority

39. Three members say yes, three members say no. One member doesn't respond at all. No decision can be reached.

40. Sometimes we can, sometimes we cannot. See the excalidraw for an example. For questions like these, its easy to make it really complicated - instead try and draw out the simplest possible scenarios.