Parallel and Distributed Computing
DD2443 - Pardis24
Exercises for Lecture 12

Name: Casper Kristiansson

October 4, 2024

Exercise 1

Question

Modify the Dolev-Strong algorithm presented in the lecture slides for Byzantine consensus using authentication so that it handles arbitrary input. The processes may also agree on a "sender is faulty" value. Give a proof that your algorithm is correct.

Answer

The Dolev-String algorithm that was present in the lecture can be modified so that it can handle arbitrary input so that a process may agree on a "sender is faulty" value. The algorithm can be designed so that if any node has detected a wrong message it will broadcast it as a faulty message. A message should be considered faulty if it is either inconsistent or wrongly signed. When the following node receives the $f + 1$ faulty message it will then make the decision that the sender of that message is faulty. By doing this we can ensure that the nodes only accept consistent, signed messages and ensuring that all messages that are either wrong or not signed properly can be considered faulty. The proof of this can be defined as:

- All correct nodes will broadcast when a faulty message/node has been identified

- The node receiving the $f + 1$ faulty message guarantees that at least one correct node can detect and identify the fault.
- Because of this all correct nodes can agree when a sender's message or the sender is faulty therefore ensuring that we have consensus.

Exercise 2

Question

Explain why the Ben-Or randomized consensus algorithm presented in the lecture slides does not work when processes just deterministically set $x = 1$ instead of choosing x randomly.

Answer

The Ben-Or randomized consensus algorithm presented in the lecture breaks when processes deterministically set $x = 1$ instead of choosing$x$ randomly. This is because the algorithm relies on randomization so that the processes do not end up in a deadlock or disagreement. If the process would always set $x = 1$ it would result in no randomness and therefore break the algorithm. The issue why this is a problem has to do with that the processes will hold specific x values where the goal is that each process might propose a value change for the other processes. But if the processes would always set their value to 1 the other processes will not be able to tell the other processes to switch to another value therefore breaking the algorithm and getting stuck in a deadlock.

Exercise 3

Question

We consider the Ben-Or randomized consensus algorithm presented in the lecture slides.

1. Show that the algorithm can tolerate $f < n/8$ crash failures, but that it is incorrect if $f \geq n/8$. *Hint: Look for a scenario for which all processes work correctly.*
2. Modify the algorithm so that it deals correctly with $f < n/4$ crash failures.

Answer

(1)

The Ben-Or randomized consensus algorithm presented in the lecture can ensure that it can tolerate $f < n/8$ crash failures. The reason why it can uphold this amount of fault tolerance is because the remaining active processes can still for a majority and therefore decide on a consensus value. The algorithm relies on that the majority of the processes can vote and therefore a failure less than $n/8$ still allows for the majority to be able to vote. But for situations where if $f \geq n/8$ it may leave too few processes being able to create a majority and therefore not being able to form and agree on a vote and therefore not upholding consensus.

(2)

For the second part where the goal is to modify the Ben-Or randomized consensus algorithm so that it can deal correctly with $f < n/4$ crash failures can be achieved by instead of constantly waiting for the majority of n-f in the proposed algorithm we can rather accept other scenarios where for example we rely on a multiple of f $n - 3f$. Making sure that the processes wait for at least $n - 3f$ proposal before processing it will allow the system to reach consistent decisions while still being able to handle $f < n/4$ crash failures. Therefore by using $n - 3f$, we can make sure that there will always be enough responses for the correct processes to prevent a deadlock.