Iowa State University, Department of Computer Engineering

Computer Science 612
Assignment 3

Ibne Farabi Shihab

December 5, 2020

1 First Problem

The transaction commit is not solvable in the given settings. At first we need to show that we can reduce consensus to transaction commit problem. In consensus, we sometimes allow fault and sometimes we do not. Here we will assume the consensus where every processor n need to agreed on a single value to come to a consensus. The formulation is same for the transaction commit problem where every processor need to agreed on commit independently before the global commit happens, otherwise there will not be any commit or the whole transaction will abort.

Here, as consensus is reducible to transaction commit, showing consensus is not solvable in an asynchronous system subject to crash failures will also prove the same for transaction commit.Before going further, we can also notice that sending opinion is same as message passing system.

The transition from bivalent to monovalent, requires a critical step which allows the transition while making a decision. But, we know critical steps can not be local and also can not be across multiple processes. Thus, transition from bivalent to univalent state is not possible which implies that the consensus can not be reached. As we reduced consensus to transaction commit problem, this also can not be possible with the given settings.

We know that, in failure free async MP system, initial state is monovalent where consensus can be reached but in case of failures the initial state is bivalent. Changing the input assignments from all 0-case to all 1-case, there must exists input assignment I_1 and I_2 which are 0-valent and 1-valent respectively. It also need to differ in the input values for a processor say p_i.

Now, if there exist a 1-crash failure tolerant consensus system,(if 1 is not possible then more than one is not possible obviously) then:

- Say, we started from I_a. If p_i fails immediately , other processors must agreed on 0 due to terminal condition, 1 in case of same scenario starting when we start from I_b. But, the first and second execution seems identical to all other processors and that is why they must have to agreed on the consensus value of 0. It is a contradiction actually. Hence, there exists at least one bivalent initial state.

2 Second Problem

Before going into the problem, lets define Test and set.

Definition: Test-and-set is a read-modify-write instruction on some binary register (let's just say that 0 and 1 are possible values) where a thread obtains the old value and writes 1.

Definition. Consensus is reached among n processors iff all n processors decide on the same value (the consistency requirement) and all processors decide on a value that was actually proposed by one of the processors (the validity requirement).

Defintion: A consensus is wait-free iff every execution call finishes in a finite number of steps.

Now follow two proof sketches.

Claim: it is possible to solve consensus in this system even if one processor crash.

Proof: Suppose that we have two processor $p_[1]$ and $p_[1]$ that need to reach consensus. We could do this by letting each processor follow the following consensus :

-Write your proposed value to Arr[i], where i is the processor id and Arr is an array of size 2.

- Perform the test-and-set instruction on some register R, initialised to 0.

If the return value is 0, you were first: return A[i]. Otherwise, you were second: return A[[i-1]].

it is trivial to verify from lectures that consensus and wait-freeness are satisfied here. Hence, our claim is true.

3 Third Problem

The claims and definition from problem 2 are hold here too.

Suppose we have three processors p_1 , p_2 and p_3 that wish to decide on values a, b and c, respectively. We will also assume that we have some valid wait-free consensus protocol that is implemented using test-and-set (and atomic reads and writes).

We can visualise the consensus process as a directed tree, like below:

The root is the state where none of the processors have 'made a move'; - The left child of a node represents the state that results from a move by p_1, the middle child represents the move by p_2, and the right child represents the move by p_1.

- A leaf node represents a state in which all processors are in finished state. A leaf node contains a value a, b or c, where the value depends on which value was decided on for that particular execution.

We know a state be multivalent if the outcome of the consensus process is not yet determined. In other words, not all possible inter-leavings of the remaining moves lead to the same result. We also know, a state be univalent when the outcome of the consensus process is determined.

Claim: The root is multivalent.

Proof: If only one processor p_i is active and the other threads lie dormant forever, then p_i will finish in a finite number of steps (guaranteed by the wait-freeness assumption) and it will decide on x (for it has only access to this value and its decision will satisfy the consensus validity requirement). So in the current set up, a, b,c are all possible outcomes.

Definition. Assume, a critical state be a state which is multivalent, with the additional property that a move by p_1 will determine a, and a move by p_2 will determine b.

Claim: There exists a critical state.

Proof: From above we know that we start in a multivalent state. Let's say, C makes no move at all. As long as either A or B does not force the tree into a univalent state, let it make a move. Wait-freeness guarantees that the tree is finite, so at some point a critical state must be encountered. Now consider a scenario where we are in a critical state. There are at least two possibilities:

1) p_1 makes its move (thereby determining a) and halts. p_2 then makes its move and halts. Next p_3 runs until it finishes, eventually deciding on a.

2) p_2 makes its move (thereby determining b) and halts. Next C runs until it finishes, eventually deciding b. A does not make a move.

Since atomic reads and writes have consensus number 1, p_1 and p_2's moves had to be test-and-set instructions on the same register (if the registers are different, then p_3 would not be able to tell the order in which p_1 and p_1's moves happened). From p_3's perspective, then, scenarios 1 and 2 are indistinguishable, so we must have that p_3 decides both a and b. But,This is impossible. Hence, it is not possible to solve.

4 Fourth Problem

Problem 1: The message complexity of synchronous message-passing model subject to Byzantine faults is $\theta(n^{f+1})$ where n is the number of processor and f is the number of failures.

Problem 2: We know that For any m, general algorithm reaches consensus if there are more than 3m generals and at most m traitors. Assuming phase king algorithm has been used the message complexity will be O($n^2 f$) for f+1 phases.

It is visible that the complexity of problem one grows faster than problem 2. Hence, we need to add overhead in problem 2 for transforming from problem 1 to problem 2. It is because we use 2 rounds in phase king which increase the overhead for the conversion. We need to add extra number of majority messages for converting from problem 1 to problem 2.

Reference

1. I answered Question 2 and 3 understanding the idea which can be found in the following link: http://www.cs.yorku.ca/ ruppert/papers/dcn.pdf

2. I answered Question 2 understanding the idea which can be found in the following link: https://www.cs.utexas.edu/users/misra/psp.dir/consensus.pdf

3. I answered Question 3 understanding the idea which can be found in the following link: https://walkccc.github.io/CLRS/Chap26/26.2/
4. I answered Question 4 understanding the idea which can be found in the following link: http://www.cse.chalmers.se/edu/year/2014/course/TIN093_Algorithms_LP1/examsolution-2014-10-29.pdf
http://people.seas.harvard.edu/ cs224/spring17/lec/lec25.pdf