CS5320: Distributed Computing, Spring 2020

Quiz2

SAI HARSHA KOTTAPALLI

CS17BTECH11036

1. The impossibility result states that

"In a fully asynchronous message-passing distributed system, in which one process may have a crash failure, it has been proved that consensus is impossible. However, this impossibility result derives from a worst-case scenario of a process schedule, which is highly unlikely. In reality, process scheduling has a degree of randomness."

To prove so:

Let T be a bivalent decision state reachable from the initial state and p be a decider process, such that action 0 by process p leads to the final decision 0(which is 0 valent), and action 1 by process p leads to the final decision 1(which is 1 valent). If p itself crashes then the system will reach some bivalent state U such that there is a decided process u. Since the algorithm must terminate, bivalent states cannot form a cycle. This implies there must exist a reachable bivalent state R, in which (1) no action leads to another bivalent state, and (2) each enabled process r is a decider. If r crashes, then no consensus is reached as there is no way to know if a message is still in transit or the decided process has crashed.

2. Given all non faulty processes propose the same value v.

Let f be the number of faulty processes.

At Phase 1.

Here, each non faulty process receives n - f votes for v in round 1.

Since in phase king algorithm we know that $n > 4f$.

We can show that $n - f > n/2 + f$.

With this condition, since majority is v, and mult > $n/2 + f$.

End of this phase every non faulty process will have v as the value.

Using the same argument, At end of Phase 2, every non faulty process will have v as the value.

Using the same argument, At end of Phase 3, every non faulty process will have v as the value.

So on ……

At end of last phase, every non faulty process will have v as the value.

Hence, the validity condition is satisfied.

3. a. The message m[0] is sent again only if the message is lost in transit/delayed or if its acknowledgment is lost in transit/delayed.

But if message m[1] is sent, this implies that the sender actually received the acknowledgement for m[0](since, this is the only way next value is incremented). And then hereby, only messages m[1] or higher order are sent by sender. since next >= 1.

Given that the channels are in FIFO, with the above reasoning, m[0] can never be sent after m[1] was sent.

b. Since m[1] is in the channel using the above reasoning for (3a), we can conclude acknowledgement for m[0] was received by sender. It is possible that due to channel delay an m[0] was delayed hence, the state of channel (m[0],0) followed by (m[1],1) is possible. Before the sender can send m[2] into the channel. it must have received acknowledgement for m[1]. This inturn implies that message m[1] was received by the receiver. Therefore, all the m[0]'s before m[1] must have already been received by the receiver or are dropped(As the channel is in FIFO).This is a contradiction, Hence, m[2] cannot be in the channel when m[0] is still in transit.

4. a. **Agreement**: Each non faulty process obtains the j th entry of its vector using the j th instance of the byzantine agreement protocol. Thus, from the agreement property of the byzantine agreement protocol, it follows that, for each j, the j th entry of the vector will be identical for all non faulty processes.

Validity: Clearly, if process Pj is non faulty, the jth instance of the byzantine agreement protocol will decide on vj - the value proposed by Pj (followed by byzantine agreement problem).

Hence, Agreement and validity conditions are satisfied.

b. **Agreement**: Clearly, all nonfaulty processes agree on all entries of their vectors(Followed by Interactive Consistency problem). Hence, majority function will evaluate to same value for all non faulty processes

Validity: The validity property of the interactive consistency protocol guarantees that if process Pi is nonfaulty, then all nonfaulty processes will decide on vi as the ith component of their respective vectors. Now, suppose all nonfaulty processes propose the same value, say v. Since a majority of processes are actually faulty. It is possible that they collude such that the majority is a value which is not the one being proposed by non faulty processes. Hence, the majority function would evaluate to a different value than v.

Example: Let there be 10 processes out of which only 3 are non faulty. Let proposed value v be “1” by nonfaulty processes. The resultant vector in IC_decide() for these nonfaulty processes due to collusion by faultycan be

<1, 1, 1, 0, 0, 0, 0, 0, 0, 1> where the last 7 are faulty. Clearly the majority function evaluated to 0 and hence validity is violated.

Hence. Agreement is satisfied but Validity is violated