CSCI/ECEN 5673: Distributed Systems
Spring 2022
Midterm Exam Review Guide
03/09/2022

1. Review all home works.
2. Review RPC details: basic methodology, issues and general idea of what unique features different RPC systems provide.
3. Given a sample message passing scenario, such HW 1 (Question 2), construct local clock and vector clocks.
4. Consider a *synchronous distributed system* with *n* processes of which up to *f* processes may fail at any time.

   (a) What is the maximum value of f to guarantee consensus among correct processes if the communication system is reliable and

   (i) processes may suffer crash failures?
   n (n-1 is also acceptable answer)

   (ii) processes may suffer Byzantine failures?
   f < n/3

   (b) What is the maximum value of f to guarantee consensus among correct processes if the communication system suffers from omission failures and processes may suffer crash failures?
   0

   (c) Suppose the communication system supports *authenticated messaging* that cannot be compromised by any process even under Byzantine failure scenario. In an authenticated messaging system, (i) a receiver can verify the identity of the sender and (ii) if the sender forwards a message it received from some other process, the receiver of the forwarded message can detect any message tampering of the forwarded message, i.e. if the sender makes any changes in message content before forwarding, the receiver of the forwarded message can detect it. What is the maximum value of f to guarantee consensus among correct processes if all processes use authenticated messaging system and processes may suffer Byzantine failures? Explain your answer.
   Consensus is possible with any number of faulty nodes.

   Each loyal node can detect when traitors send inconsistent values. If the coordinator is loyal, the loyal lieutenants will get the same values from the coordinator and from other lieutenants (otherwise they will detect message tempering). Thus agreement can is achieved. If the coordinator is traitor, the lieutenants may receive conflicting orders from the coordinator, which they can detect on receiving conflicting information in forwarded messages from other lieutenants.

5. Consider a synchronous distributed system that has reliable message delivery, maximum message delay of *M* time units, and minimum message delay of *m* time units. Further maximum drift rate between the client and the server clocks is *d* time units per hour. Assume that processing time is negligible. Christian's probabilistic clock synchronization algorithm is used to synchronize clocks.

(a) Assume $M = 20$, $m = 17$ and $d = 2$. What are the error bounds within which a client may synchronize her clock with a server's clock?

Error bounds are ±((T1-T0)/2 – Tmin).
The maximum value of T1-T0 is 40 time units.
So the error bounds are ±3 time units.

(b) Again assuming $M = 20$, $m = 17$ and $d = 2$, how often (largest time interval) should the client synchronize her clock to always stay within 50 time units of the server's clock? Explain your answer.

When a client synchronizes her clock, it may be up to 3 time units off. Every hour, this clock may drift by 2 time units. So, after 23 hours, the clock may be off by as much as 49 time units. So, the largest time interval for synchronization is 23 hours.

6. Suppose you have implemented two different fault-tolerant data structures, FTStack and FTQueue using Raft. FTStack provides the standard stack implementation and tolerates up to two server failures. FTQueue provides the standard queue implementation and tolerates up to two server failures. An application developer needs an atomic operation that pops an item from an FTStack object and pushes that item on to an FTQueue object. She implements this operation as follows:

```
FTStack *st; FTQueue *que;
...
int x = st->sPop( );
que->qPush(x);
```

(a)] Explain what problem this implementation may run into.
There may be a failure after the item is removed from stack and before it is inserted in the queue. This will leave the system in an inconsistent state.

(b) Explain how you will fix this implementation to address the problem you identified in (a).
Note: you don't need to provide the detailed code.
Make the code segment with the two statements a transaction, e.g. by using a 2-phase commit protocol.

(c) Suppose another atomic operation the application developer needs is retrieve a copy of the top item of a FTStack object (without popping the item) and push that copy to a FTQueue object. She implements this operation in a similar fashion:

```
FTStack *st; FTQueue *que;
…
int x = st->sTop( );
que->qPush(x);
```

Does this implementation suffer from a similar problem that you identified in (a). Explain your answer.

No this doesn't suffer from that problem. This is because sTop operation doesn't change the state of the stack. So, a failure after this statement doesn't affect the system state.

7. Consider a multicast message passing system where a message sent by a process is received by every other process in the system. There are no unicast messages in this system.

(a) What changes if any would you make in the Lamport's logical clock algorithm that we built for a unicast messaging system in class to work for this multicast messaging system? Explain your answer.

No change is needed.

(b) Define FIFO order message delivery for this system.

If two messages, m_1 and m_2 are sent from the same process, such that send(m_1) → send(m_2), then every process that delivers both m_1 and m_2 must deliver m_1 before m_2.

(c) Define causal order message delivery for this system.
For any two messages, m_1 and m_2, if send(m_1) → send(m_2), then every process that delivers both m_1 and m_2 must deliver m_1 before m_2.

(d) Illustrate via diagram to show that messages may not be received in causal order even if the underlying system guarantees FIFO ordering delivery.
See HW solution

8. **(a)** What is virtual synchrony in group membership protocols? Explain the difference between virtual synchrony and extended virtual synchrony.

   See Lecture Set Eight

   (b) Describe a scenario under which a 2-phase commit protocol will block. How does a 3-phase commit protocol prevent such blocking?

   See Lecture Set Nine

9. What is the key limitation of replicated state machine approach to build fault tolerant systems? Explain how a primary-backup approach addresses this problem. (Answer in at most 3-4 lines)

   Replicated state machine approach doesn't work if the computation is non-deterministic. Primary-backup approach solves this problem by having only primary server perform the computation and then updating the states of the backup servers, so the execution at the primary server dictates the correct execution.