15-440/640 Distributed Systems
Midterm SOLUTION

Name:
Andrew ID:

October 17, 2019

- Please write your name and Andrew ID above before starting this exam.
- This exam has 16 pages, including this title page. Please confirm that all pages are present.
- This exam has a total of 100 points.

Question	Points	Score
1	13	
2	26	
3	12	
4	12	
5	15	
6	12	
7	9	
8	1	
Total:	100	

True/False

1. Circle whether each statement below is *true* or *false*. ***ALSO***, give a one to two sentence reason for your answer.

   Each correct answer is worth 3 points (1pt for the T/F and 2pts for the reason).

   (a) (3 points) [True/False] Protocols such as 802.11 (WiFi) never retransmit packets since the end-to-end argument suggests that reliability must be done at the end-points.

   Solution: False - Low level mechanisms to support these functions are justified only as performance enhancements. 802.11 retransmits packets a limited number of times after collisions to ensure good performance.

   (b) (3 points) [True, False] Over a wide area network it is better to use TCP to implement Voice-over-IP telephony rather than UDP given the potential for packet losses.

   Solution: False - UDP is still going to be better for the VoIP app since VoIP is real time and the delays caused with TCP retransmision would make things worse. What's more, VoIP is loss-tolerant.

   (c) (2 points) [True, False] In the LSP protocol from Project 1, the server should send out all the pending messages when its Close() is invoked and cannot return until it receives all the ACKs because these messages can still get lost. **Just answer T/F, no justification needed.**

   Solution: True.

   (d) (2 points) [True, False] In P1, when Close() has been invoked, the server can stop sending heartbeats, because sending the last pending messages takes less time than the heartbeat interval. **Just answer T/F, no justification needed.**

   Solution: False.

   (e) (3 points) [True, False] A RAID system using spinning hard disks is likely to have its throughput bottlenecked by the CPU cost of calculating the parity block.

   Solution: False. Disks are comparatively slow and calculating XOR is very fast.

Short Answers

2. In the following, keep your answers brief and to the point (i.e., 2-3 sentences).

   (a) (4 points) Can you implement an RPC system using UDP connections, instead of TCP? Answer yes/no and explain why or why not in at most two sentences.

   Solution: Yes. It doesnt matter what transport layer protocol you choose for the RPC system. You can choose to implement over an unreliable network connection and rely on RPC system to detect and tolerate the failures.

   (b) (4 points) A developer decides to use RPCs to remotely invoke the function SumLinkedList(ListItem *head). What problem might they encounter?

   Solution: RPC systems don't handle pointer structures well. The marshalling system would need to figure out how to follow the linked list structure and include all the needed memory locations.

   (c) (4 points) Andi makes a small addition to the beginning of a large file. Since they added data, all the existing data has been moved by a small amount. In most systems, every block or chunk of the file would have to be updated to reflect the new content. How does the LBFS file system ensure that most of the file chunks remain the same and avoid transferring the remaining data?

   Solution: LBFS uses Rabin fingerprints, which are based on the contents within a small part of the file, to determine chunk boundaries. As a result, the chunk boundaries after the beginning of the file will remain unaffected.

(d) (4 points) You have a new disk that allows writes in 64 byte quantities. For your transactional database system, however, you need to be able to write 512 byte sectors atomically. Describe how you can achieve the effect of an atomic write of 512 bytes in this situation. It's OK to use more than 512 bytes.

Solution: Best solution involves writing a checksum. They could also pick a transaction ID and write it at the start and end.

(e) (4 points) Consider a two-phase commit system with four nodes, the coordinator and three workers, A, B, and C. The coordinator crashes after sending its `VoteCommit?` message to the workers, and then returns to life having missed all of the responses.

Can the system commit? [Circle One] Yes No

Explain briefly why.

Solution: Both answers are acceptable:
No. The system cannot commit, as one of the missed replies could have been a VoteAbort. Commiting in this case would be incorrect.
OR
Yes. The coordinator could use the gossip protocol and commit if it finds that all workers have voted to commit.

Can the system abort? [Circle one] Yes No

Explain briefly why.

Solution: Yes. The system can safely abort. No worker will have committed yet, so abort is the safe option.

(f) (6 points) Nodes A and B wish to time synchronize. The link from A to B takes 40ms, and the link from B to A takes 20ms but these numbers are not known to the computers. They synchronize using Cristian's algorithm in one round. Node A's time is 500 and Node B's time is 632. Node A starts the protocol.

(a) At the completion of the protocol what time does Node A believe it is?

Solution: It takes 40ms for the packet to reach B, so B timestamps it as 632+40 = 672. The response reaches node A with an RTT of 60ms, so node A sets its time to 672+60/2=702.

(b) What, if any, error is in this value and why?

Solution: At the end of the protocol Node B will think it is 692. The error is 10ms. It arose because of the asymmetry in the links.

Why Would Adults Share Chopsticks Like This?

3. There is a classical problem in concurrency known as the Dining Philosophers. The problem is as follows: Five philosophers sit around a circular dining table. Each philosopher has a bowl of rice in front of them, and there is a single chopstick exactly in between every two philosophers. Each philosopher alternately thinks and eats. A philosopher needs a pair (i.e., chopstick to their left AND right) of chopsticks to eat. When the philosopher is thinking, they put down their chopsticks.

   Given that this is an example of a concurrent system with a need for synchronization, three important design goals that a potential should address, when there are **NO FAILURES** in the system are:

   - No Deadlocks - everybody can eat and are not waiting on each other.
   - No Livelocks - everyone can eat and are not just picking and dropping chopsticks
   - Fairness - everyone gets a chance to eat at some point after a bounded amount of waiting.

   (a) (4 points) Consider a potential solution to this problem:

   ```
   Step 1: think until the left chopstick is available; when it is, pick up;
   Step 2: think until the right chopstick is available; when it is, pick up;
   Step 3: when both chopsticks are held, eat for a fixed amount of time;
   Step 4: then, put the right chopstick down;
   Step 5: then, put the left chopstick down;
   Step 6: repeat from the beginning.
   ```

   Does this solution address all of the desirable properties of concurrent systems that we have listed above? If not, which one is violated and give an example of when it is violated?

   Solution: No. The above solution suffers from a deadlock since each philospher could pick up the chopstick to the left of them and be waiting for the chopstick on the right which will never become available. In this case, the system reaches a deadlock state and no progress is made.

(b) (4 points) Assume that we change the solution outlined in part (a) above such that each philospher is asked to wait 5 minutes after acquiring the left chopstick to get the right chopstick and if they cannot get it within that time, they let go of the left chopstick and sit idle for 5 minutes. What desirable property of concurrent systems is still being violated? Give an example of when it is violated.

Solution:
This updated solution could still cause resource starvation (or livelock) since its possible that all the philosphers come to the table at the same time, pick up the left chopstick at the same time, wait 5 minutes, drop the chopstick, wait another five minutes, pick up the left chopstick again. So, while a deadlock is avoided it causes a potential livelock.

(c) (4 points) What is a simple modification to the prototol in part (b) above that can address the property being violated?

Solution: To avoid the livelock case, the algorithm could be changed in one of the following ways:

- Have the philosphers wait for a random time (0 to n minutes) after dropping the chopstick (or even picking up the chopstick in the first place) before trying again. This would lead to the system continuing forward and avoiding the deadlock and the livelock.
- Have only one philosopher pick up the right chopstick first, followed by the left one (all the rest will pick up the left chopstick first as usual). This breaks the cyclic dependency which leads to the livelock, and allows the system to proceed.
- Have odd numbered philosophers pick up the left chopstick first and even numbered philosophers pick up the right chopstick first. Again, this breaks the cyclic dependency which leads to the livelock, and allows the system to proceed.

Note that while (2) solves the livelock problem, it **does not lead to a fair system**.

The Multiplied Mutexer

4. Consider a variation on the central coordinator algorithm for mutual exclusion. Instead of one coordinator, suppose we have two coordinators, such that the algorithm will continue to operate even if at most one coordinator fails. The coordinators are connected with a synchronous channel with maximum delay of D seconds. Communication between the coordinators and client processes is asynchronous. When a process needs to enter the critical section, it sends a request to both the coordinators. The process may enter the critical section when it receives an OK message from either of the two coordinators.

   Help define the protocol the coordinators use to communicate between each other.

   Name the coordinators P and S. P is the primary and S is the secondary. For naming purposes, assume they maintain a lock-owner variable to remember who has the lock, if held, which is nil otherwise.

   Assume that P is required to send a heartbeat message to S every T seconds.

   Ensure that your protocol works properly if the primary had granted a lock, and then died, that the secondary still knows that the lock is held and by whom.

   Define your protocol by answering the following four questions. If any of your operations involve sending a message from P to S or S to P, please define in that answer what the receiver should do with the message they receive.

   (a) (3 points) When P receives a lock request from client C, what is the sequence of operations it performs?

   Solution:

   - P checks that the lock is available, otherwise it enqueues the request and returns (this line should be exectued atomically);
   - P sends a note to S saying ”lock granted to C”;
   - P sets lock-owner=C;
   - P replies ”lock-granted” C.

   Note that it's important for P to notify S before replying to C, otherwise it might happen that P dies inbetween messaging C and messaging S. Then S could grant the lock to a different client, which would violate mutual exclusion.

   (b) (3 points) When S receives a heartbeat message from P, what does it do?

   Solution: Record last_heartbeat=current_time().

   (c) (3 points) How does S conclude that P has died?

Solution: If it has not heard from S within D + T seconds, it can conclude that P has died.
Note that because D is only an upper bound on the network delay, we cannot conclude anything after T seconds: for all we know, it could be that a packet took 0 seconds to go from P to S, and the next packet (T seconds later) took D seconds.

(d) (3 points) When S receives a lock request from client C, what does it do?

Solution: If S believes the primary is alive, it puts the request in a queue of pending requests Q, and waits.

If S believes the primary has died, it puts the request at the end of its queue and handles requests in its queue.

Handling a request operates as normal: If the lock is already held it leaves it in the queue, stops working, and waits for a lock release. Otherwise, grant the lock to C (update the variable and send a message).

Dropbox's Marketing Takes Over the Company

5. Dropbox has discovered the secret to perfect computers and networks. None of their servers crash and their network will never get partitioned. They use Quantum Magic to make messages arrive *instantaneously* (faster than light). The magic network includes their servers and all client devices. They promise that: (1) all file changes will be immediately visible to each connected user (2) Any user can access (view or modify) any file on any of their devices at any time.

   (a) (3 points) Assuming their breakthrough is real, can they deliver on these promises? Explain why or why not in a few sentences with reference to the CAP theorem.

   > **Solution:** Yes. The technological change Dropbox found has eliminated the possibility of a network partition. The CAP theorem precludes having both availablity and consistency *because* the network can become partitioned. No partitions? No problem.
   >
   > Some answers justified that the end-user device could still fail, as the question didn't specify the behavior of end-user devices, and therefore there could still be potential consistency problems. We accepted this answer with full credit.
   >
   > Many incorrect answers said "no, you cannot have partition tolerance". This answer received 0 points, because the problem was explicit that this was something magical being brought in.

   (b) (3 points) Dropboxs magic network does not cover Pittsburgh, and clients there must use normal Internet connections. To make users happy there, Dropbox uses their normal mechanism of caching files on device. They use a mechanism to allow clients to access (view or modify) those files. They say the mechanism they use can allow the client to have an exclusive lock on a file while handling the case where a client crashes forever. What is that mechanism, and explain briefly how it works.

   > **Solution:** Leases! The client is allowed to have an exclusive lock on the file for a certain amount of time. After that time, the lock is revoked automatically, so that a failed client that cannot reach the server will lose its lock.
   >
   > We accepted with full credit any answers that explained locks with timeouts. Answers that only said locking received partial credit, as such an answer is incomplete: It can prevent inconsistency but a client that died with a lock may prevent the system from functioning.

   (c) (3 points) Explain the consequence this change (allowing clients in Pittsburgh) has on the promises they made for part a. Can they still meet them? Why or why not?

Solution: No. Now the network is again subject to partitions, which means they must give up either consistency or availability.

(d) (3 points) Dropbox decides to allow Pittsburgh-to-Pittsburgh networking. Node A can allow node B to directly fetch a file from its local cache instead of going through the dropbox servers. When node A does send the file to node B and node B wants to modify it, what agreement must A and B have to ensure that the file access remains safe under the mechanism you described earlier?

Solution: A must give the lease to B (and stop accessing it itself!). If B renews the lease there needs to be some way of communicating the ownership transfer to Dropbox's servers.

Many incorrect answers said that B must send its writes through A. This solution does not work, as B's writes may not be allowed if the lease has expired, resulting in at least transient inconsistency between the nodes in the system.

(e) (3 points) [True, False] The AFS distributed file system improves scalability by keeping no per-client state. Justify your answer with a 1-sentence explanation.

Solution: False – A client must still register with the server to request that the server sends it a callback when the file is modified, which is stateful.

Why Can't We All Just Agree?

6. You just graduated from CMU and are hired as a Paxos expert. Your colleague, who didnt take 15440, keeps suggesting changes to your companys Paxos implementation, and its your job to analyze them and often rebuke them. For this problem we want two properties from a consensus algorithm:

   - Correctness: The consensus is not reached for more than one value
   - Liveness: The system never gets stuck (it never gets to a state where it will never be able to reach consensus)

   For each of the following variants of single-decree Paxos, indicate which properties it has:

   (a) (4 points) Every time they receive a message, nodes flip a coin and drop it if the coin comes up heads. Does this satisfy (Circle the right choice):

   - Correctness and Liveness
   - Only correctness
   - Only liveness
   - Neither

   Explain your answer in at most two sentences. Seriously, keep it brief.

   Solution: Solution: Correctness and Liveness. Since messages are only dropped (and not changed), this will not impact correctness. Also, with some probability, no message gets dropped for long enough that Paxos can complete (think geometric distribution).

   (b) (4 points) Once they have accepted a proposal, nodes never change their mind (and reject later proposals with different values regardless of proposal number).
   Does this satisfy (Circle the right choice):

   - Correctness and Liveness
   - Only correctness
   - Only liveness
   - Neither

   And explain. Again, keep it brief:

   Solution: Only correctness. System might get stuck in a 33%/33%/33% split and never move on.

   (c) (4 points) The prepare phase is only used for finding the highest sequence numbers, but proposer may still use their value instead of what they get from running the prepare phase if they feel strongly about it.
   Does this satisfy (Circle the right choice):

- Correctness and Liveness
- Only correctness
- Only liveness
- Neither

And explain. You know the drill, please keep it brief:

Solution: Only liveness. A consensus can be overwritten. Example: Node 3 becomes the proposer after A and B have accepted a value v. It then proposes value v'. But because A and B have already accepted the value, it may have been committed.

Primary Backup Backup - Backup!

7. Dawn Bovik is designing a new primary-backup system - they want to assign two backups to the primary, marked as backup-A and backup-B. Their replication mechanism works as follows:

   - If the primary fails, backup-A is promoted to be the new primary.
   - If backup-A dies or is promoted, backup-B is marked as backup-A.

   To make sure the system works with slow and unreliable networks, Bovik decides to replicate only a single copy of each operation as follows:

   - The primary will forward each operation at first to just backup-A.
   - If the RPC fails, then only in that case does the primary forward to backup-B.
   - If that RPC fails, the primary tries backup-A again, and so on, until the RPC to one of them succeeds.

   (a) (3 points) Can this system tolerate 2 server failures? Justify in 1 sentence

   Solution: No (1pt for mentioning no), in the general case there may be only one copy of each operation so it may get lost. We also wanted you to mention that the primary cannot backup if there are neither of the backups available, so will keep trying.

   (b) (3 points) Can this system tolerate disconnection of backup-A from the network (in the absence of any other failures)?

   Solution: Ans. Yes (1pt for mentioning yes). this protocol allows the primary to continue quickly in the face of slow/unreliable connectivity to backup B .

(c) (3 points) Bovik realizes this design has a serious consistency flaw - provide an example of scenario where this system will return an inconsistent result under failures that a paxos system should be able to handle (i.e., one of the three nodes fails).

Solution: Ans. The client sends put(x,99) to the primary; the primary forwards the put only to backup-B; the primary fails and backup-A becomes the new primary; now a get(x) wont return 99. You had to mention two aspectsto get full credit: (a) Backup A and Backup B don't have the same contents so they are inconsistent. (b) Also mention that the Backup-A be becomes primary on the failure of the primary, then the system will return an inconstent result.

Anonymous Feedback

8. (1 point) Tear this sheet off to **receive points**. We'd love it if you handed it in either at the end of the exam or, if time is lacking, to the course secretary.

   (a) Please list one thing you'd like to see improved in this class in the current or a future version.

   (b) Please list one good thing you'd like to make sure continues in the current or future versions of the class.