15-440/640 Distributed Systems
Midterm SOLUTION

Name:

Andrew ID:

October 16, 2017

• Please write your name and Andrew ID above before starting this exam.

• This exam has 18 pages, including this title page. Please confirm that all pages are present.

• This exam has a total of 100 points.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
True/False

1. Indicate 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] The checksum portion of an IP packet allows the receiver to both detect and correct a single-bit error in the packet.

Solution: False. (Checksums can be used to detect errors, but not to correct them.)

(b) (3 points) [True, False] With Lamport clocks, if $e_1$ “causes” $e_2$ (i.e. $e_1$ is causally before $e_2$), we are guaranteed that the Lamport clock times for these events are such that $L(e_1) < L(e_2)$?

Solution: Yes. Lamport clocks always increase between causally related events.

(c) (3 points) [True, False] A system with a low MTTF ($<$ 1 day) will always also have low availability.

Solution: False - if the MTTR is very small (e.g. failures last for a random millisecond every hour)
Short Answers

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

(a) (3 points) How does NTP account for the fact that a server may take some time after receiving a time query to respond to the client?

Solution: The server sends the client both the time at which it received the query and the time at which it sent the reply, allowing the client to calculate the exact time the query and the response were in transit.

4 points for mentioning that the time at which the query is received and the response is sent is included in the response to the client. 2 points if the answer vaguely mentions that the server records timestamps.

(b) (4 points) Some of the 440 students decide to create a network to exchange messages during lecture using a network of ultrasonic links. These links are a bit slow: bandwidth is only 1Mbps. Srini and Daniel try using the network. Daniel is sitting in the back of the room, 100ft away from Srini. Sound travels at 1000ft/sec. The links have limited range and the path between Daniel and Srini takes some hops through intermediate routers. Srini measures the RTT for a minimum size packet to be 320ms. Compute the time it takes to deliver a 1500 byte packet from Srini to Daniel.

Solution:

\[ 1.5k \times \frac{8}{1M} = 12ms + 160ms = 172ms \]

0 points if the 100ft data is used for calculation, as it has no relevance to the problem at all.
-2 if RTT is not divided by 2.
-1 for minor calculation errors, bit vs byte, M vs K, etc.

(c) (3 points) If Srini wants to ensure that a file transfer to Daniel uses the full network bandwidth, what is the minimum window size (in bytes) should he use for his transfer?

Solution: \[ 1Mbps \times 320ms = 320kb = 40KB \]

(d) (4 points) In very large distributed systems, it is often impractical to implement fault tolerance using synchronized checkpoints as many requests would queue up. Another way to implement checkpoints is independent checkpointing. What is the disadvantage of independent checkpointing?
Solution: The domino effect, aka, cascaded rollback. No single checkpoint leads to a consistent state and we roll back further and further.
Full points if answer mentions hard to find consistent state, leading to higher roll back.
No points reduced if domino effect or cascaded rollback terms are not used.

(e) (6 points) Another way to implement fault tolerance is write-ahead logging (WAL). List the name of each step (pass) that is performed during recovery using WAL. Also explain what is done in each pass (one sentence each).

Solution: Analysis Pass: Reconstruct TT and DPT (from start or last checkpoint) Recovery Pass or Redo Pass: Replay log forward, make updates to all dirty pages Undo Pass: Replay log file backward, revert any changes made by transactions that had not committed (use PrevLSN). Three points for approximate names, and three points for descriptions.
1 point each for the (approximately) correct names. 1 point each for the de-
scriptions.
(f) (4 points) X writes a program on a single-core machine to complete a task in bounded time, or abort it if time is exceeded. Assume that the following program compiles. Is this a good implementation?

```go
package main

import (
    "fmt"
    "time"
)

// The time limit for task (in seconds)
const TIMEOUT_IN_SECS = time.Duration(2);
var resultChan = make(chan int);

func task() {
    // Tons of work, and the result is stored in res
    resultChan <- res
}

func main() {
    var startTime = time.Now()
    go task()
    for {
        select {
            case res := <- resultChan:
                fmt.Println("The result is", res)
                return;
            default:
                if time.Now().Sub(startTime) >= TIMEOUT_IN_SECS * time.Second {
                    fmt.Println("Timeout. Abort the task.")
                    return
                }
        }
    }
}
```

**Solution:** No. The spinlock in main routine will consume great number of CPU resources and significantly reduces the efficiency of completing the task. This is a very common mistake for students implementation in P1, and anyone who has pay attention to that should be able to answer it. 1 point for just saying no.
| No additional points for saying that the goroutine will not exit when the main program exits after a timeout. |
| No points for saying that the timeout is too small. |
| Full points for not mentioning spin lock, but saying that the main loop will take up CPU cycles and the task won’t get enough. |
One algorithm to achieve distributed mutual exclusion is the Lamport Mutual Exclusion algorithm. The figure below shows an instance of this algorithm with three processes \( p_0, p_1, \) and \( p_2 \) (with respective IDs, 0, 1, and 2).

Fill into the gaps the *totally ordered Lamport* timestamps. Assume that the first Lamport timestamp is \( L = 0 \) for \( p_0 \), \( L = 1 \) for \( p_1 \), and \( L = 2 \) for \( p_2 \).

**Solution:**

**Correct Variant 1**

\[
\begin{array}{cccc}
\text{p}_0 & \text{L}=0 & \text{L}=6 & \text{L}=9 \\
\text{L}=1 & \text{L}=10 & \text{L}=13 & \text{L}=21 \\
\text{p}_1 & \text{L}=5 & \text{L}=8 & \text{L}=14 \\
\text{L}=2 & \text{L}=17 & \text{L}=20 & \text{L}=23 \\
\text{p}_2 & \text{L}=2 & \text{L}=5 & \text{L}=8 & \text{L}=17 & \text{L}=20 & \text{L}=23 & \text{L}=21 \\
\end{array}
\]

\[\text{req(5)} \quad \text{req(8)} \quad \text{ack(p}_1\text{)} \quad \text{ack(p}_0\text{)} \quad \text{release}\]

\[\text{release}\]

p2 enters critical section

**Correct Variant 2**

\[
\begin{array}{cccc}
\text{p}_0 & \text{L}=0 & \text{L}=6 & \text{L}=9 \\
\text{L}=1 & \text{L}=10 & \text{L}=13 & \text{L}=24 \\
\text{p}_1 & \text{L}=5 & \text{L}=8 & \text{L}=17 \\
\text{L}=2 & \text{L}=20 & \text{L}=23 & \text{L}=26 \\
\text{p}_2 & \text{L}=2 & \text{L}=5 & \text{L}=8 & \text{L}=17 & \text{L}=20 & \text{L}=23 & \text{L}=21 \\
\end{array}
\]

\[\text{req(5)} \quad \text{req(8)} \quad \text{ack(p}_1\text{)} \quad \text{ack(p}_0\text{)} \quad \text{release}\]

\[\text{release}\]

p2 enters critical section

-1 for making a single error and then the error propagates -1 for using given timestamp * 3 + pid as new timestamp. Full points for using 10*i + pid as timestamp. No points reduced for missing the numbers on the arrows.
(h) (4 points) Another algorithm to achieve distributed mutual exclusion is due to Ricart and Agrawala. Which of the messages from the previous picture would be exchanged during the Ricart and Agrawala Algorithm?

**Solution:** Only four out of the six messages: the two request messages and the two ack messages.
Full points for this answer.
3 points for converse answer.
2 points for partially correct answer (eg. just request messages sent or just ack messages sent)
The next two questions rely on your understanding of the LSP (Live Sequence Protocol) that you implemented as a server-client message transferring protocol in P1.

(i) (4 points) Is it possible for a client to get two ACKs with the same sequence number? If so, describe an example that would cause this to happen. If not, explain why it can’t happen.

Solution: It is possible. Imagine the client sends a data message right before an epoch event fires. The client notices it has an unacknowledged data message, and resend the data message. Both data messages reach the server, and the server sends two ACKs in response. Another solution is that the underlying network itself duplicated the message. Both solutions are accepted. Half credit for saying it is possible but not correctly explaining why.

(j) (4 points) At epoch 3, the client sends a data message to the server that does not get ACK’d. Give the numbers of the next 5 epochs during which the client will attempt to retransmit the data message, using the same strategy as you did in your project. Assume MaxBackOffInterval is 5.

Solution: The client retries at epochs 4, waits one epoch, retries at 6, then 9, 14, and 20. Half credit if they’re off by one, or forget that we initially retry after 0 and 1 epochs, before then doubling the amount of epochs.
3. Consider converting the following three local functions “square”, “squareAll”, and “saveSquare” into RPCs.

```go
func square(int a) {
    return a*a
}

func squareAll(intList list) {
    outlist = new intList
    for(entry in list) {
        outlist.append(entry*entry)
    }
    return outlist
}

func saveSquare(int a) {
    file = open("/project/squares.txt")
    write(file,a*a)
}
```

(a) (3 points) The “square” function is the easiest of the functions to convert into a RPC. However, even it can cause issues. Consider the client application running on an x86 architecture while the RPC server is running on a SPARC architecture. Describe one issue that may arise from this CPU architecture difference.

**Solution:** 3 points were awarded for answers explaining endianness, data type overflows or bit representation differences.

(b) (3 points) Give one reason why a RPC implementation of the “squareAll” is more challenging than the “square” function?

**Solution:** Because one has to copy and serialize the full list over the network. -1 if marshalling complications or deep copying is not specified.
(c) (3 points) Give one reason why a RPC implementation of the “saveSquare” is more challenging than the “square” function?

Solution: Because it has side effects on the local/remote file system and another function might want to read the same file, but is executed on another machine.

3 points: Local vs. remote file system namespace differences or side effects (non-idempotency) of the “saveSquare” function.
2 points: Consistency issues between multiple client writes on the server.
1 point: Any other valid explanation such as unclosed file descriptors, etc.
RAID and Reliability

4. You have joined AIT (A superfamous institution) and have been asked to design a storage-array for old archives. You use super cheap disks with a Mean-Time-To-Failure (MTTF) of 10,000 hours ($\simeq 1.1$ years). The drives have the following performance characteristics:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTTF</td>
<td>10,000 hours</td>
</tr>
<tr>
<td>Sequential read/write speed</td>
<td>100 MB/sec</td>
</tr>
<tr>
<td>Capacity</td>
<td>2 TB</td>
</tr>
<tr>
<td>Seek time</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

(a) (3 points) Assume you have built a RAID 1 (“mirrored” – each disk has a copy of all data) system using two disks. Without any humans around to replace dead drives, what is the expected mean time to data loss of two disks using RAID 1? (Keep your answer simple, using the simplified model of MTTF calculations from the lectures).

Solution:

\[
\frac{1}{2} \text{MTTF} + 1 \text{MTTF} = 1.5 \text{MTTF} = 1.5 \times 10,000 \text{hours} = 15,000 \text{hours}
\]

1 point awarded if the approach is correct but the final answer is wrong.

(b) (3 points) When a disk fails, it takes about 5 hours to rebuild the array by copying all data onto a new disk. Using the simplest model, what are the chances of the second disk dying during this rebuild?

Solution: 5 hrs / 10,000 hours = 1/2000
Seagate has just announced new super cheap and fast drives called SCF drives. However, writing data to the SCF drives is the primary cause for failure (i.e. they suffer from "wear-out"). Assuming a normal write workload with mostly small writes, these drives have a amazing MTTF of 504 years. Higher write workloads cause the MTTF to drop in proportion to the workload increase (e.g., twice as many writes makes the MTTF 1/2 as long)

(c) (3 points) You begin using these SCFs by building a RAID 4 array (striping + parity disk). Assume a configuration with 7 drives + 1 parity drive. What issue might this layout create?

**Solution:** The parity disk would get 7 times as many writes. This would cause it to fail quickly. So the array would then lose data when the second drive failed – i.e. in about MTTF/7 = 72 years

More accurately:

\[ \text{MTTF/14} \] [the first failure is MTTF/7 on the 7 data drives and MTTF/7 for the parity]

\[ + \]

.5 * MTTF/7 50/50 that parity drive failed first so now a failure of a data drive or.

\[ + \]

.5 * (MTTF/13) if data disk died first

\[ M/14 + M/14 + M/26 = 33M/182 = 0.18 \text{ MTTF} \]

A valid explanation is sufficient for full points, calculations are not necessary.

(d) (3 points) Suppose you try to address this issue by building a RAID 5 array (striped parity) using these SCFs. What would the MTTDL be for this modified array?

**Solution:** Each of the 8 drives would get 1/8 extra write traffic. They should then fail in 8/9 the time = 448 years.

With 8 drives, the MTTDL would be 448/8 + 448/7 = 56 + 64 = 120 years

2 points for the correct equation but using 504 years as the MTTF. 1 point for any valid calculation.
Distributed File Systems

5. Andrew plans to deploy a distributed file system on campus. To choose between AFS and NFS, Andrew decided to look at several design aspects of both systems.

(a) (4 points) On a single UNIX machine, if some process B reads a block of a file after it has been updated by another process A, the copy of the file block B reads will include A’s updates. What are the guarantees (consistency semantics) when A and B are running on different machines and the block is part of a file mounted using NFS on both machines?

**Solution:** In NFS, cached data is updated only periodically (mentioned as 30 seconds in the lecture slides). Thus, it is possible that B could read old data for a while after A has finished updating it. The semantics are those of “weak consistency”.

4 points for mentioning client periodically (30 - 60 secs) updates server or weak consistency.

(b) (4 points) The AFS solves the above problem in (a) by using state information it maintains at the server. What state is kept on the server, and how is the state used to solve the problem?

**Solution:** An AFS server keeps track of which clients have copies of particular files. Thus, when one client writes data and closes the file so that the data is flushed to the server, the server contacts each of the clients that have cached copies of the file and tells them to invalidate the file via *callbacks*.

2 points for mentioning that server keeps track of which clients have copies of particular files. 2 points for callback from server.

(c) (4 points) Different workloads can affect the performance of any systems. 1) What does AFS assume about the frequency of write/write and write/read sharing? 2) What kind of workload can be problematic for AFS?

**Solution:** AFS assumes that write/write and write/read conflicts are rare. It is problematic for AFS if an application appends information, periodically, to a log. These little log writes, which add small amounts of data to an existing large file, are quite problematic for AFS.

2 points for the AFS assumption for the frequency of write/write and write/read sharing. 2 points for problematic example.
6. Arianne is the tech lead at XCheck and has been tasked with creating a distributed database for accounts and payments. In order to support millions of customers, the data of different accounts will live on different servers. Arianne is trying to decide between the Two Phase Commit protocol (2PC) and the Paxos consensus protocol (Paxos).

(a) (1 point) For 2PC, Arianne is concerned about what happens if the coordinator crashes. Can a coordinator crash lead to an inconsistent database state, violating atomicity (yes/no)?

**Solution:** No.

(b) (4 points) For 2PC, assume that there are two participating servers A and B, and both send “VoteCommit” to the coordinator. The coordinator, sends out the “DoCommit” message to A, but crashes immediately before sending a message to B. When do A and B commit their respective changes to their local database?

**Solution:** A commits immediately after receiving the “DoCommit” message. In classical 2PC (without the gossip protocol discussed in class), B waits until the coordinator recovers. After recovery, the coordinator reads its log (which says that it decided to commit) and resends the “DoCommit” message to B. Then B commits.

In 2PC with the gossip protocol, B asks A whether A has received a “DoCommit” message. Once A answers B, B can also commit.

One point for when A commits. Three points for either the classical solution (without gossip) or the gossip protocol solution.
(c) (4 points) For Paxos, Arianne is concerned about having to wait for a majority of acceptors. She considers an implementation of Paxos where a proposer waits for less than a majority of acceptors to answer OK to a Prepare or an Accept message before it proceeds to executing the next steps. Describe a scenario where Arianne's implementation may not correctly match Paxos' service guarantees.

**Solution:** The Paxos correctness guarantee is that only a single value is chosen (and one that has been proposed). With Arianne's proposal of not waiting for the majority, two proposers might get two different values chosen. This leads to an inconsistent distributed database.

Scenario: Consider a partitioned network with 50 servers each, out of 100 servers in total. If proposer A is in one partition and proposer B in another, both can get their value chosen as they wait for less than a majority.

Other scenarios: Delayed messages, slow responses, etc.

One point for correctness guarantee. One point for “two values” get chosen, two points for a scenario.

(d) (3 points) For Paxos, how many messages will be exchanged to update the balance of a single account. Assume that there are 100 servers.

**Solution:** We want to see a lower bound and an upper bound.

**Lower bound.** Paxos algorithm is defined such that proposer always sends its messages to all nodes (100), even if some are down. But some of the nodes may be down, so at minimum we received 51 replies. So, if 49 nodes are down, and if there are duelling proposers, we will send 100+51+100+51 = 302 messages. If we count the “commit” message (which is not part of the core Paxos protocol) to everyone, another 100 messages can be added, for a total of 402 messages. Both 302 or 402 are correct.

**Upper bound.** If there are dueling proposers, an infinite number of messages might be sent in Paxos (e.g., if one server is slow to respond, and another servers assumes its role).
Anonymous Feedback

7. (2 points) 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.