Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.5840 Distributed System Engineering: Spring 2025

Exam I

Please write your name on the bottom of each page. You have 80 minutes to complete this quiz.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, write down any assumptions you make. Write neatly. In order to receive full credit you must answer each question as precisely as possible.

You may use class notes, papers, and lab material. You may read them on your laptop, but you are not allowed to use any network. For example, you may not look at web sites, use ChatGPT, or communicate with anyone.

Name:

Gradescope E-Mail Address:

Grade histogram for Exam 1

$$\begin{aligned} \text{total} &= 68 \\ \text{max} &= 68 \\ \text{median} &= 55 \\ \mu &= 54.85 \\ \sigma &= 6.7 \end{aligned}$$

I MapReduce

The MapReduce paper (*MapReduce: Simplified Data Processing on Large Clusters*, by Dean and Ghemawat) says in Section 3.1 that the intermediate key space is partitioned among the R reduce tasks using hash(key) mod R.

Thea is running the word-count MapReduce job (pseudo-code in the paper's Section 2.1) on a cluster with 10 worker machines. M is 20 and R is 40. There are no failures, the network is reliable, no machines are slower than expected, and there is no competing work on any of the machines or networks involved. The Map input is divided into 20 pieces of 16 megabytes each.

1. [5 points]: By mistake, the hash(key) function Thea is using with MapReduce always returns 1. What effect will that have on the execution of the word-count job, compared to using a well-behaved hash function? Circle the single best answer.

A. the job will produce incorrect final output

B. 10 times as much total CPU time will be needed for Reduce phase

C. 10 times as much total wall-clock time will be needed for Reduce phase

D. 40 times as much total CPU time will be needed for Reduce phase

E. 40 times as much total wall-clock time will be needed for Reduce phase

F. the job will never complete

Answer: C. The total amount of computation is unchanged, but it's all done by one worker rather than divided up in parallel among 10 workers. B is not correct because the total amount of work doesn't change; the only thing that the hash function changes is which worker does the work.

Name: _____

II VMware FT

The paper *The Design of a Practical System for Fault-Tolerant Virtual Machines* by Scales et al. describes a method for avoiding split-brain using shared storage. Instead of using the shared storage's test-and-set, Ben implements test-and-set using the `kvsrv` server from Lab 2 as follows:

```
func test-and-set(clnt *tester.Clnt) bool {
   val, version, err := clnt.Get("lock")
   if err != rpc.OK {
       return false
   }
   if val == "set" {
      return false
   }
   if err := clnt.Put("lock", "set", version); err == rpc.OK {
      return true
   }
   return false
}
```

The `clnt.Put` and `clnt.Get` are RPCs that invoke the server's `Put` and `Get` methods. You can assume that Ben has implemented the `Put` and `Get` methods correctly.

The initial value of "lock" is the empty string.

When the primary or the backup suspect that the other one has crashed, they invoke `test-go-live`, each with their own RPC client `clnt`:

```
func test-go-live() {
    for true {
       if test-and-set(clnt) {
           go-live()
           return
       }
    }
}
```

Name: _____

The network may lose, delay, or duplicate a few messages, but most messages will be delivered. The computers (primary, backup, and kvsrv server) do not fail.

2. **[5 points]:** What statements about Ben's implementation are true? (Circle all that apply)

   A. Both the primary and backup may observe `test-and-set` returning true in `test-go-live`, and "go live", resulting in split brain

   B. The key/value server may never store "set" for the "lock" key

   C. The primary and backup may spin forever in `test-go-live`, retrying `test-and-set`, because it may never return true

   D. If all RPCs succeed with no timeouts while running `test-go-live`, either the primary or the backup will observe true from test-and-set, but not both

Answer: C is true: the first Put may change lock to set, but the reply may be lost; a re-send will return ErrMaybe (since the version won't match); so neither primary nor backup will ever see rpc.OK from Put. D is true: if there are no timeouts (i.e. no packets are lost) the first Put to arrive at the kvsrv will succeed, and the sender will get rpc.OK.
A is false, because `Put` is conditional and only one can set `val` to "set". B is false, because most messages will be delivered and thus eventually a `Put` will succeed in setting the lock.

Name: _____

III Linearizability

Alyssa is experimenting with a linearizable put/get key/value storage service. Unlike Lab 2, her key/value service has no versions; put calls look like put(key, value).

Alyssa has two clients. Client C1 executes this:

```
t = get("x")
put("x", t + 1)
```

At about the same time, client C2 executes this:

```
t = get("x")
put("x", t * 2)
```

Before either client starts, the value for key "x" in the storage system is 10. Both clients' calls complete without error. There is no other activity involving the storage system, and there are no failures.

Suppose the history of the execution, in the style of Lecture 4, with values omitted, looks like this:

```
C1: |--Rx?--|   |--Wx?--|
C2: |--Rx?--|   |--Wx?--|
```

3. [5 points]: After both clients have finished, what could the resulting value of x be in the storage system? (Circle all that apply)

A. 10

B. 11

C. 20

D. 21

E. 22

Answer: 11 and 20. Both C1's read and C2's read see the initial value of x (10), so C1 writes 11 and C2 writes 20. The writes are concurrent, so linearizability allows either write to appear to execute last, and thus provide the final value.

Name: _____

Alyssa resets the value of "x" to 10, and re-runs the two client programs. This time, the execution history looks like this:

```
C1: |--Rx?--|   |--Wx?--|
C2:      |---Rx?---|   |--Wx?--|
```

4. [5 points]: After both clients have finished, what could the resulting value of x be in the storage system? (Circle all that apply)

A. 10

B. 11

C. 20

D. 21

E. 22

Answer: 11, 20, and 22. 22 is possible if C2's read sees C1's write.

Name: _____

IV GFS

Consider GFS as described in *The Google File System* by Ghemawat *et al.*

5. [5 points]: Which statements about GFS are true? (Circle all that apply)

A. GFS ensures linearizability of client operations by allowing clients to read from chunk replicas.

B. The primary server of a chunk ensures that Append operations are executed exactly once.

C. A chunk server uses 64 Mbytes of disk space for each chunk.

D. Leases help ensure that each chunk has only one primary.

Answer:
A is false, because GFS allows reading chunks from backups, which may have not seen the last update to a chunk, violating linearizability. B is false because if an Append fails, the client retries the Append, which the primary executes, causing some Appends to be execute twice. C is false; the paper's Section 2.5 says that chunks are stored as Linux files and are extended only as needed, with disk space allocated lazily; this means that if only a few bytes of a chunk are written, only that part of the Linux chunk file will consume disk space. D is true; Section 3.1 says that the coordinator grants a chunk's lease to just one of the replicas, and only grants the lease to a different replica if the lease expires.

Name: _____

V Raft

Refer to Ongaro and Ousterhout's *In Search of an Understandable Consensus Algorithm (Extended Version)*.

6. [10 points]: Which statements about Raft are true? (Circle all that apply)

- **A.** If a follower receives an AppendEntries RPC from the leader and the follower's term matches the one in the RPC, then the `prevLogIndex` in the RPC must be equal to or higher than the follower's `lastApplied`
- **B.** Raft is optimized for the case that term switch happen frequently
- **C.** Raft guarantees that a leader in term t is leader in term $t + 1$
- **D.** If a leader sends the command in log index i on the apply channel, the leader must have persisted log index i
- **E.** If a follower crashes in a term and quickly reboots, it remembers who it voted for before the crash
- **F.** The leader's `matchIndex` for a peer is always equal to or smaller than the leader's `nextIndex` for that peer.
- **G.** A candidate who becomes leader sends out AppendEntries to all followers to suppress further elections
- **H.** If Raft doesn't use snapshots, a crashed follower will send all committed log entries on the apply channel after it reboots, even ones that it sent before the crash

Answer: D, E, F, G, H are true.
A is false, because an AppendEntries RPC from the leader may be delayed and arrive after later AppendEntries RPCs that bump up `lastApplied`; when the follower processes the first RPC, the `prevLogIndex` may be smaller than its `lastApplied`.
B is false, because the authors believe terms change infrequently and therefore don't think the fast-backup optimization is necessary.

Name: _____

7. [10 points]: Which of the following bugs causes a Raft implementation to violate the safety properties listed in Figure 3? (Circle all that apply)

A. A deadlock in a follower

B. A follower who starts an election very quickly

C. A partitioned leader who on rejoining updates its term to the new leader's term and sends AppendEntries for commands in its log with the new term

D. A race condition in the follower's implementation that causes two followers to send different commands on the apply channel for log index i

E. A candidate that forgets to vote for itself

F. A follower who appends a log entry to its log even if the term in the AppendEntries is smaller than its own and who then sends the log entry on the apply channel

G. A follower that forgets to implement the rollback optimization presented at the end of section 5.3

H. A leader who always sends only one entry in an AppendEntries RPC to a follower

Answer: A, B, E, G, and H are examples of what are called "liveness" bugs: these bugs don't cause wrong behavior but may prevent any progress. A: A deadlock in the follower may cause Raft to not make forward progress at some point (e.g., if the follower is necessary to form a majority). B may prevent a leader from being elected but it doesn't violate the safety properties. E is another variation of B. G and H may cause Raft to run slowly but that doesn't violate the safety properties.
C, D, F, on the other hand, are "safety" bugs that cause incorrect behavior that violates the safety rules of Raft of Figure 3.

Name: _____

VI Lab 3A-3C

George is implementing Raft as in Lab 3A-3C. Eager to test his implementation, George runs a git pull to get the latest changes from the 6.5840 staff. The latest changes introduce a new test, `TestMiniReElection`, which tests whether Raft can re-elect a leader after a single network partition.

```
func TestMiniReElection(t *testing.T) {
  servers := 3 // initialize three servers

  ...

  // wait for a leader to be elected; get the leader's index
  leader1 := ts.checkOneLeader()

  ts.g.DisconnectAll(leader1) // disconnect leader1 from other servers

  // wait for a new leader to be elected; get the leader's index
  leader2 := ts.checkOneLeader() // ***
}
```

`ts.checkOneLeader()` repeatedly polls only the *connected* servers until one of the connected servers returns that it is a leader. If it cannot find a leader within 5s, it returns a timeout error.

Unfortunately, there is a bug in the Raft test infrastructure. When `leader1` is disconnected, `leader1` can still send RPCs to the other servers but not receive responses from the other servers. George runs `TestMiniReElection`, and finds that the test fails at the line marked with "***" with the timeout error "expected one leader, got none".

8. [5 points]: Assume George's Raft implementation is completely correct, and that the network is reliable. Briefly explain why the buggy test infrastructure causes George's implementation to fail the new test.

Answer: All the peers will continue to receive leader1's heartbeat AppendEntries RPCs, which will prevent them from ever starting an election.

Name: _____

VII ZooKeeper

Refer to *ZooKeeper: Wait-free coordination for Internet-scale systems*, by Hunt, Konar, Junqueira, and Reed, and to Lecture 9.

Alyssa runs a ZooKeeper service with a ZooKeeper leader and multiple followers. Alyssa has three ZooKeeper client programs, P1, P2, and P3:

```
P1:
    s = openSession()
    if create(s, "/leader", "one", flags=ephemeral) == true:
        print "P1 starting as leader"
        _, version = getData(s, "/x", watch=false)
        setData(s, "/x", "one", version)
        _, version = getData(s, "/y", watch=false)
        setData(s, "/y", "one", version)

P2:
    s = openSession()
    if create(s, "/leader", "two", flags=ephemeral) == true:
        print "P2 starting as leader"
        _, version = getData(s, "/x", watch=false)
        setData(s, "/x", "two", version)
        _, version = getData(s, "/y", watch=false)
        setData(s, "/y", "two", version)
        print "P2 done"

P3:
    s = openSession()
    sync(s, "/")
    x = getData(s, "/x", watch=false)
    y = getData(s, "/y", watch=false)
    print x, y
```

Initially, znode "/leader" does not exist, znode "/x" exists and contains the string "empty", and znode "/y" exists and also contains the string "empty".

The ZooKeeper calls in Alyssa's code are all synchronous. The ZooKeeper client call create() is exclusive, returning false if the file already exists, and true if it was able to create the file. The programs might end up talking to different ZooKeeper followers.

Name: _____

Alyssa starts P1, waits until she sees it print "P1 starting as leader", then (on a different computer) starts P2. Just at this point in time, P1's network connection starts to become slow and unreliable, so that sometimes it delivers packets, sometimes not. Alyssa sees that P2 prints "P2 starting as leader", and after a little while "P2 done". P2's network connection is reliable and fast.

After Alyssa sees "P2 done", she runs P3.

9. [5 points]: What output from P3 could Alyssa see? (Circle all that apply)

A. one, one

B. two, two

C. one, two

D. two, one

Answer: Only two, two. We know P1's session must have terminated, because Alyssa saw P2 print "P2 starting as leader," which could only have happened if ZooKeeper deleted P1's ephemeral /leader file. So P2 will only start reading and writing data after P1 is guaranteed to have stopped writing (since ZooKeeper terminated its session). So P1 and P2's activities won't be intermixed; P2 runs strictly after P1. So both P2's sets will succeed. P3 starts after P2 finishes, and P3 calls sync(), so P3 will see P2's writes.

Name: _____

VIII Distributed Transactions

Alyssa has a database that supports serializable transactions. Records "x" and "y" both start out containing the value 1. Alyssa starts three transactions at the same time:

```
T1:
  BEGIN-X
    temp1 = get("x")
    temp2 = get("y")
    put("x", temp1 + temp2)
  END-X

T2:
  BEGIN-X
    temp1 = get("y")
    put("x", temp1 * 2)
  END-X

T3:
  BEGIN-X
    put("y", 3)
  END-X
```

`BEGIN-X` marks the start of a transaction, and `END-X` marks the end. All three transactions commit and finish. There are no aborts, deadlocks, or failures. There is no other activity in the database.

When Alyssa looks at record "x" in the database after the transactions complete, she sees the value 5.

10. [5 points]: Briefly explain how the value 5 could have resulted from these transactions.

Answer: The database system could have executed the transactions one at a time, in the order T2, T3, T1.

Name: _____

IX Spanner

Refer to *Spanner: Google's Globally-Distributed Database*, by Corbett *et al*.

11. [5 points]: Suppose you only wanted to support read-write transactions (not read-only and not snapshot reads). You want therefore to eliminate all of the Spanner mechanisms that are not needed for read-write transactions. Which of these techniques can be eliminated? (Circle all that apply)

A. commit wait (Sections 4.1.2, 4.2.1)

B. safe time (Section 4.1.3)

C. deadlock avoidance (Section 4.2.1)

D. assignment of timestamps to read/write transactions (Section 4.1.2)

Answer: A, B, and D. The time-stamp mechanism is only needed for read-only transactions. Read-write transactions are made serializable and externally consistent by Spanner's two-phase locking and two-phase commit; the correctness of read-write transactions thus does not rely on time-stamps.

Name: _____

X 6.5840

12. [1 points]: Which lectures/papers should we definitely keep for future years?

- MapReduce 122
- RPC, Threads, and Go 80
- Primary-Backup Replication / VMware FT 64
- Linearizability 84
- Raft 125
- GFS 85
- ZooKeeper 79
- Distributed Transactions 68
- Q+A on Lab 3A/3B 50
- Spanner 52

Name: _____

13. [1 points]: Which lectures/papers should we omit?

- MapReduce 2
- RPC, Threads, and Go 16
- Primary-Backup Replication / VMware FT 29
- Linearizability 18
- Raft 1
- GFS 14
- ZooKeeper 19
- Distributed Transactions 14
- Q+A on Lab 3A/3B 49
- Spanner 41

14. [1 points]: What can we do to improve the course?

Labs	**Count**
Better debugging tools	15
Fine	12
Implement things other than Raft	10
More/better tests	9
Better lab specs/common bugs	8
More lab guidance	7
Make labs independent/release lab solutions	7
More time to do labs	5
More time to learn Go/Go tutorials	5
Shorter labs	2
More system design practice	1
Extra credit for labs	1
More difficult labs	1

Continued on next page.

Name: _____

Lectures	**Count**
More context for papers	7
More about distributed systems applications	5
More code examples during lecture	4
Better lecture note formatting	4
Better FAQ	4
Order lectures more logically	3
More lecture QAs	3
Eliminate lecture QAs	3
Better lecture visualization tools	3
Answers to lecture QAs	3
Have lectures be different from the papers	2
Be more engaging	2
Focus on big ideas during lectures	2
More guest lectures	1
Free candies at lecture	1
Misc	5

Logistics	**Count**
More OH/more TAs during OH	22
Record lectures	6
Turn up the lecture audio	2
Recitations	2
Better support for projects	2
Don’t use Gradescope	1
Better website	1
Weigh labs more	1
Give out Go plushies	1
More collaboration	1
Professor OH	1

Exams	**Count**
Review session	3
Weight exams less	2
More practice questions	1
No exams	1
Bigger midterm test room	1

End of Exam I

Name: _____