*Department of Electrical Engineering and Computer Science*

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

## 6.5840 Distributed System Engineering: Spring 2025

# Exam II

Please write your name on the bottom of each page. You have 120 minutes to complete this exam.

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.

The maximum number points available is 76.

## Gradescope E-Mail Address:

**Name:** _____

# I FaRM

Consider the following statements about FaRM as described in *No compromises: distributed transactions with consistency, availability, and performance*. For each statement, circle True or False.

**1. [8 points]:**

**True / False** : Short leases are important for FaRM, because FaRM must reconfigure to remove a failed replica of a region before FaRM can continue to use the region.

**True / False** : For small messages RDMA performs 4× better than RPC because the CPU is a performance bottleneck for RPCs.

**True / False** : To obtain write locks FaRM uses RDMA writes so that the destination machine's CPU doesn't have to do any work on a lock request.

**True / False** : For the TATP workload the median latency on the left end of the graph is 9 microseconds (see Figure 7), rather than the 19 microsecond mean commit latency mentioned in Section 6.3, because not all operations update multiple rows.

**Name:** _____

# II Chardonnay

Answer these questions with reference to *Chardonnay: Fast and General Datacenter Transactions for On-Disk Databases* by Eldeeb *et al.*

**2. [3 points]:** Section 6.1 says that all of a read-write transaction's writes are tagged with a version containing the same epoch number. For example, if a read-write transaction executes at around the time the Epoch Service changes the epoch from 20 to 21, either all the transaction's updated records should have a VID prefixed with 20, or all should be prefixed with 21, but not a mix of the two. Explain briefly why having the *same* epoch number in all the written items' versions is important for correctness.

**3. [3 points]:** Section 6.1 describes how a transaction chooses a version ID (VID): it is the current epoch, with a counter appended. Suppose, instead, that version IDs were just the current epoch (with nothing appended). Briefly explain how this change could cause snapshot reads to yield incorrect (non-serializable) values.

**4. [3 points]:** Section 8 says that the locks for a transaction are acquired one at a time using a "chain" technique. It would be faster if the client sent out all the lock requests in parallel. Briefly explain why sending a transaction's lock request messages in parallel would be a bad idea.

**Name:** _____

# III Grove

Consider "Grove: a separation-logic library for verifying distributed systems" by Sharma et al. For each statement, circle True or False.

**5. [8 points]:**

**True / False** : Because any replica in Grove can serve a Get request, Grove achieves higher throughput for Get's than for Put's (see Figure 19), but sacrifices linearizability for Get's.

**True / False** : If a response to a Get is lost, the Grove `exactlyonce` library will resend the Get, which the server executes again and the second response may contain a different value than the first (but lost) response contained.

**True / False** : If the Go code for vKV had a bug that caused a backup to lose the effect of a Put after a crash, then Grove's specification and proof would catch this bug.

**True / False** : If the `configservice` had an infinite loop that caused `Reconfigure` to never return, then this bug would be caught by Grove's specification and proof.

**Name:** _____

# IV DynamoDB Transactions

With respect to *Distributed Transactions at Scale in Amazon DynamoDB* by Idziorek *et al.*, imagine that a read-only transaction (TransactGetItems) and a read-write transaction (TransactWriteItems) execute concurrently. The read-write transaction updates multiple items stored on different storage nodes; the read-only transaction reads the same set of items. Serializability requires that the read-only transaction see all of the items as they were before the read-write transaction, or all as they are after the read-write transaction (but not a mix of before and after).

**6. [5 points]:** For the above scenario, which mechanisms help to ensure this all-before or all-after property? Refer to the design as described in the paper's Sections 2 and 3 (but not Section 4). Circle all that apply.

**A.** The item.timestamp $<$ input.timestamp check in Listing 3.

**B.** Multiple time-stamped versions stored for each key.

**C.** Log sequence numbers (LSNs).

**D.** Two-phase locking.

**E.** item.ongoingTransactions

**Name:** _____

# V AWS Lambda

Consider the paper, and guest lecture about, *On-demand container loading in AWS Lambda* by Brooker et al. For each of the following statements, indicate whether it is true or false.

**7. [8 points]:**

**True / False** : AWS Lambda is attractive to customers because the customer can spawn many lambdas in response to a load spike without having to provision machines in advance.

**True / False** : Replication of chunks in the AZ-level cache is important to ensure that chunks are not lost forever when a cache node fails.

**True / False** : Erasure coding of cached chunks helps improve tail latency, because a worker can reconstruct a chunk without having to download all stripes of the chunk.

**True / False** : The convergent encryption scheme described in Section 3.1 helps protect against an attacker who compromises a worker and attempts to read containers of any customer.

**Name:** _____

# VI Lab 4

Ben Bitdiddle is implementing the `rsm` package from Lab 4. He uses a shared reply table to implement communication between the `rsm`'s `Submit()` method and the reader goroutine. The reply table maps a completed operation's `id` to its reply from `DoOp()`. In addition, `rsm` contains a `nextId` field that is initialized to 0 and is incremented by 1 each time `Submit()` is called.

His code takes the following steps:

**A.** The kvraft server calls `rsm.Submit()` with a client request `req`.

**B.** `Submit()` increments `rsm.nextId`, packs the request into a new operation `Op{id: rsm.nextId, req: req}`, and calls `rsm.rf.Start()` with the `Op`. If `rsm.rf.Start()` indicates that the current peer is the Raft leader, it then waits for the reply table to contain a reply under the key `Op.id` (the one passed to `Start()`).

**C.** When the `rsm`'s reader goroutine receives an `ApplyMsg` from the `applyCh`, it calls `DoOp()`, then populates the reply table with the result from `DoOp()` under key `Op.id`.

**D.** When `Submit()` sees the `Op.id` key in the reply table, it deletes the `Op.id` key/value pair from the reply table and returns the corresponding reply.

Assume that all omitted parts of Ben's design are correct.

**8. [5 points]:** Ben notices that his implementation can result in incorrect behavior. Explain why.

**Name:** _____

# VII Ray

Consider the following Ray program, which creates a `sqrt_task` task for each number in the list `mylist`. The creation yields a DFut and the caller waits for the tasks to complete by calling `get` on each future. The code is as follows:

```
# A call to sqrt_task yields a DFut
@ray.remote
def sqrt_task(n):
  # sqrt is a python function, which returns the square root of its argument
  return sqrt(n)

@ray.remote
def sum_task(f):
  # sum is a python function, which takes a future and returns the sum
  l = get(f)     # collect the list l
  return sum(l)  # return the sum of the numbers in list l

# A call to sqrt_list_task yields a shared DFut
@ray.remote
def sqrt_list_task(n_list):
  # start tasks and collect futures
  l = [ ]              # list holding DFuts
  for i in n_list:     # iterate over list of numbers
    l.append(sqrt_task(i))

  r = [ ]
  for f in l:
    r.append(get(f))   # collect the result

  return r  # return a SharedDFut for r

# invoke sqrt_list_task with a large list of numbers, sum, and print result
f = sqrt_list_task(mylist)
s = sum_task(f)
print(s)
```

**Name:** _____

Assume Ray behaves in the way described in *Ownership: a distributed futures system for fine-grained tasks* by Wang et al., and Ray is running on a cluster of computers.

For each of the following statements, indicate whether it is true or false.

**9. [8 points]:**

**True / False** : a Ray worker may start running `sum_task` before `sqrt_list_task` has finished

**True / False** : the driver that invokes `sum_task` receives the list with square-rooted numbers from the worker that ran `sqrt_list_task`.

**True / False** : the driver is the owner for each future that `sqrt_task` returns.

**True / False** : the driver is the owner for the shared future returned by `sqrt_list_task`.

**Name:** _____

# VIII SUNDR

Consider the straw-man design in the paper *Secure Untrusted Data Repository (SUNDR)* by Li *et al*.

In the straw-man design, a client asks the server to append a "fetch" operation to the history after reading the history. Each client also remembers the most recent operation it has appended to the history. One reason for these mechanisms is to prevent the server from first showing some operations to the client, and then later hiding those operations.

Suppose one modified the SUNDR straw-man protcol in three ways. First, eliminate "fetch" operations, so that the history only includes modify operations. Second, have each client remember the last entry in the most recent history the client obtained from the server; call this remembered entry ELAST. Third, each client checks that ELAST is present in the next history it obtains from the server. With these modifications, a client can still detect a situation in which the server first shows an operation to a client, and then omits that operation in a later history shown to the same client.

**10. [5 points]:** It turns out this modification is a bad idea. Explain briefly how this modification would allow a malicious server to violate fork consistency.

**Name:** _____

# IX Bitcoin

*Bitcoin: A Peer-to-Peer Electronic Cash System*, by Nakamoto, mentions in Section 4 that the cryptographic hash of a valid block must start with a certain number of zero bits. Assume that the hash algorithm is SHA-256, which returns a 256-bit hash.

**11. [3 points]:** You are trying to mine a new block. The required number of zero bits is seven. You set the block's 32-bit nonce field to a random value, and compute the SHA-256 hash of the block. What's the probability that the first seven bits of the hash are zeros? Circle the one best answer.

**A.** 1/2

**B.** 1/7

**C.** 1/128

**D.** 1/256

**E.** 1/249

**F.** $1/(2^{32})$

**Name:** _____

Ben runs a Bitcoin node. A few hours ago Ben's node learned about block B747, which is a valid block. Ben sees a transaction T27 in B747 that pays some money to a certain public key, signed by the correct private key. Ben would like to steal the money involved in T27. He modifies his copy of block B747 so that the payee's public key in T27 is Ben's own public key. He doesn't change anything else in B747. He modifies his Bitcoin node software to announce the block to other nodes as if it were a valid block.

**12. [3 points]:** Which of the following will cause other Bitcoin nodes to decide that Ben's B747 is invalid? Circle all that apply.

**A.** The "Prev Hash" field in the next block in the chain doesn't refer to Ben's B747.

**B.** Other peers will already know about the real B747.

**C.** The "Prev Hash" field in Ben's B747 isn't valid.

**D.** The hash of Ben's B747 won't start with enough zeroes.

**E.** The signature in T27 in Ben's B747 isn't correct.

Now Ben is designing a new crypto-currency system, identical to Bitcoin, except with a different agreement scheme to resolve forks in the block-chain: instead of the longest fork winning, nodes compute the hash of the last block in each fork, and the fork with the lowest last-block hash value wins. Ben reasons that all nodes will compute the same hashes, and thus all nodes will agree about which fork wins.

**13. [3 points]:** Why is Ben's fork-resolution idea a disaster? Explain briefly.

**Name:** _____

# X PBFT

Consider the PBFT protocol as described in the paper *Practical Byzantine Fault Tolerance* by Castro and Liskov.

For each of the following statements, indicate whether it is true or false.

**14. [6 points]:**

**True / False** : Assume all replicas of a PBFT-replicated system are running the same software and the software has a bug. If an attacker exploits this bug in all replicas to control each replica, then the attacker can commit arbitrary operations in the log.

**True / False** : If an attacker controls the primary of a PBFT-replicated system, then the attacker can commit arbitrary operations in the log.

**True / False** : Honest replicas must include in a VIEW-CHANGE the messages already prepared so that a new primary cannot omit already-committed operations.

**15. [2 points]:** If there are 10 machines in our system, and at most 2 machines are malicious, how many REPLY messages does a client need to receive before it knows its operation was executed? (Circle best answer)

**A.** 1

**B.** 3

**C.** 4

**D.** 6

**E.** 7

**Name:** _____

# XI 6.5840

**16. [1 points]:** Which lectures/papers should we **omit** in future years?

- FaRM
- Chardonnay
- Grove
- Memcache at Facebook
- DynamoDB
- AWS Lambda
- Ray
- SUNDR
- Bitcoin
- PBFT

**17. [1 points]:** Porcupine, the linearizability checker used in the labs, comes with a visualizer that displays in a web browser the entire history of client and tester operations. Circle the one closest to your experience with the visualizer.

- I haven't used it for debugging.
- I've tried using it, but it wasn't helpful in my case.
- I've read the visualization, and it occasionally helped me identify the issue in my code.
- I've read the visualization, and it often helped me identify the issue in my code.
- I've added my own annotations, and it enhanced my debugging experience.
- Other (please briefly describe your experience):

**18. [1 points]:** Do you have any feedback for us about 6.5840?

# End of Exam II

**Name:** _____