Problem Set 3

15-440/15-640 Distributed Systems Spring 2026

Assigned: Thursday, March 19, 2026

Due: Thursday, March 26, 2026 at 11.59 pm

Submission procedure:

- Create a **.pdf** of your answers and upload to `Gradescope`.
- Here are some tips to make your submission easy to read and to grade. Remember, the easier you make this, the less likely we are to make grading errors. Following these guidelines will help us to focus on the technical content of your answers rather than trying to understand what you have written.
    - Don't handwrite your answers. Use LaTeX or Google Docs or some similar input mechanism. If you use LaTeX, a template can be found on the course web page.
    - Put the answer to each question on a separate page.
    - Carefully tag your pdf pages to each question on `gradescope`. You can use the SHIFT key to select multiple pages and associate them with a single question.
- Assume SI notation
    - 1 KB = 10^3 bytes, 1 MB = 10^6 bytes, 1 GB = 10^9 bytes
    - 1 Kbps = 10^3 bits per second (bps), 1 Mbps = 10^6 bps, 1 Gbps = 10^9 bps
    - but a byte is still 8 bits (not 10 bits) :-)
- **Remember that you have a limit of 2 grace days totaled over all four problem sets.** You can use at most one of those grace days per problem set. Although Gradescope does not track grace days, we will. Exceeding your grace days will result in a zero grade for that problem set.

Question 1 (20 points)

As a member of a small start-up, you're working on building and deploying a web application that wraps various LLMs to provide market research to aspiring day-traders. Your web application consists of three tiers: (1) a frontend that handles browser interaction; (2) the backend that handles the majority of the business logic and LLM querying; (3) a database that records user information and various transaction details. Due to some of the team's earlier design decisions and some preferences from the C-suite, each tier has the following constraints:

- Frontend: Must consist of machines of the same size only.
- Backend: At least 50% of the computation cycles have to occur on one machine, the remaining percentage of computation cycles can be distributed across multiple machines.
- Database: To ensure data integrity, this tier must run on a single machine.

Benchmarking results show that the resources used (CPU, RAM, disk) for each tier of the application scales linearly with the number of users. Specifically, the resource usage on a given machine follows an equation of the form y = Ax + B, where x is the number of clients on that machine and A and B are constants defined below.
Note: Gops = Giga operations / second

Resource Requirements per Application Tier						
Tier	CPU Demand (Gops)		RAM		Disk	
	A	B	A	B	A	B
Frontend	0.02 Gops	0 Gops	75 MB	0.1 GB	50 MB	0.1 GB
Backend	0.04 Gops	0 Gops	0.1 GB	4 GB	0.1 GB	0.2 GB
Database	0.02 Gops	0 Gops	55 MB	2 GB	0.5 GB	1 GB

The following machine types are available for use in each tier of the application.

Resource Availability by Machine Type					
Type	Cores	Per-core CPU Speed	RAM (GB)	Disk (GB)	Hourly Cost ($)
Small	2	2.0 Gops	8	128	$0.04
Medium	4	2.5 Gops	32	256	$0.30
Large	8	3.2 Gops	64	512	$1.10
X-Large	16	7.0 Gops	256	2048	$2.80

A. Suppose there is a constant load of 500 concurrent users. What is the most cost-effective configuration of machines for each tier of the application? Show your work.

B. For the configuration calculated in part (A), what is the maximum number of concurrent users that can be supported? Identify (with explanation) which tier of the application is the bottleneck for this particular configuration.

C. Instead of renting machines, your start-up considers purchasing them at a liquidation sale. The heavily-discounted cost of each machine type is shown below. What is the cost for your start-up to purchase the configuration that you identified as optimal in (A)? Relative to renting that configuration, how long would it take to break even for that purchase?

Instance Type	Purchase Cost
Small	$800
Medium	$3800
Large	$15,500
X-Large	$65,000

D. As your company grows, demand during peak hours (2pm to 2am) jumps to 750 concurrent users. What is the cheapest configuration that satisfies the new peak requirements? Assuming that rented infrastructure may be scaled in and out immediately in response to changing demand, what is the new breakeven time period for renting versus buying? You may assume that the purchase configuration must be the same as the rent configuration required for peak load.

Question 2 (20 points)

You are training a new ML model that utilizes a novel transformer architecture. You decide to use a distributed training algorithm that uses several GPUs simultaneously to train faster. You are forced to use old GPUs that are beginning to show their age and occasionally reset during operation. All GPUs must remain operational for the entire duration of the training job; if any GPU resets, the entire job will crash and must be restarted from the beginning.

Your early experiments show that:

- Each GPU fails independently
- The time to failure of each GPU follows an exponential distribution. Such a distribution is “memoryless” — the probability of failing in the next moment is 1/M, where M is the mean of the distribution. That is, future probabilities do not depend on past history. Mathematically, if X is exponentially distributed, it says that $P(X > x + k \mid X > x) = P(X > k)$.
- The mean time between failures (MTBF) for each GPU is 10 minutes.

You run the training job using five GPUs. Show your full work and reasoning when answering the questions below.

A. Suppose a GPU has not reset for 8 minutes. When do you expect this GPU to experience a reset next? Why?

   Hint: Remember the memoryless property of the exponential distribution.

B. What is the expected time until some GPU fails in a long running training job?

C. If a job takes T minutes to complete on perfectly reliable GPUs of the same type that you are using (no restarts at all), how many attempts will it take on average to run successfully on your flaky hardware? You may assume that failure detection and restart happen instantaneously. You may give your answer as an algebraic expression, without reducing it to simpler form.

   Hint: remember that the entire job must be restarted from scratch on each attempt.

D. Using your answer to (C), obtain numeric values of the number of attempts on your flaky hardware for the following values of T
   a. T = 5 minutes
   b. T = 10 minutes
   c. T = 60 minutes

E. From your answers to (D), what qualitative insight about running long jobs on flaky hardware do you obtain?

Question 3 (18 points)

For each of the following failure scenarios, state and explain whether it is fail-fast or Byzantine. If it is a Byzantine failure, explain how to make it fail-fast.

A. The following pseudo-code for a transfer of money:

```
def update_balance (account_number, change):
    if account_number not in accounts:
        print("Account does not exist")
        return -1

    new_balance = accounts[account_number] + change

    if new_balance < 0:
        print("Insufficient funds")
        return -1

    accounts[account_number] = new_balance
    return new_balance

def transfer_money (buyer, seller, amount):
    update_balance(buyer, -amount)
    update_balance(seller, amount)
    print("Transfer completed")
```

B. On a new video call service, to handle temporary internet disconnections, the system keeps displaying the last received video frame from a participant until the connection is restored.

C. A café uses a QR-code ordering system where customers place and pay for orders from their phones. During peak hours, if the kitchen is overloaded, the app times out and shows “Order failed, please try again” and does not charge the customer.

D. When a printer’s ink cartridge is nearly empty, the printer still shows a “printing” status on the screen and feeds the paper normally. However, due to the low ink level, the printer only prints the top half of the page, leaving the bottom half blank. The printer does not display any warning or error message indicating that the ink is insufficient.

E. The Find-My-Friends app is designed to display the current location of your friend’s phone when they choose to share their location with you. However, if your friend’s phone dies, the app cannot update the location; it no longer shows up in your list of friends on the map.

F. You are at a Sephora store and redeem 100 of your 700 loyalty points to receive a Sephora reward, a new facial moisturizer. The cashier completes the transaction. However, later when you open and reload the Sephora app, it still shows that you have 700 points in your account.

Question 4 (14 points)

For the pseudo-code segments below, state whether an ACID property is violated. If so, state which property is being violated and why; also identify a bug fix with explanation in two to three sentences.

A.
```
// Copy a tree structure to a new location and delete it from the previous location.
01. tree_node *move_tree(tree_node *original_node, tree_node *new_node) {
02.     if (original_node != NULL) {
03.         move_res = copy_node(original_node, new_node);
04.         if (!move_res) {
05.             abort_move(new_node);
06.             return original_node;
07.         }
08.         child_count = 0;
09.         child_nodes = get_children(original_node, &child_count); // Stores num children in child_count
10.         for (int i = 0; i < child_count; i++) {
11.             new_child_node = create_child_slot(new_node);
12.             child_res = move_tree(child_nodes[i], new_child_node);
13.             if (!child_res) {
14.                 abort_move(new_node);
15.                 return original_node;
16.             }
17.         }
18.         delete_res = delete_node(original_node);
19.         if (!delete_res) {
20.             abort_move(new_node);
21.             return original_node;
22.         }
23.     }
24.     return new_node;
25. }
```

B.

```
// Handle flight reservations for an airline
// Recovery operations (if any, from an earlier crash) are completed before this function is called
01. def flight_reservation_manager():
02.     reservation_state = get_state_from_disk()
03.     while server_active:
04.         reservation_request = receive_request()
05.         txn_handle = new_transaction()
06.         res = attempt_request(reservation_request, reservation_state)
07.         if res:
08.             write_to_log(txn_handle, reservation_request)
09.             send_confirmation(reservation_request)
10.             commit_transaction(txn_handle)
11.         else:
12.             send_failure(reservation_request)
13.             abort_txn(txn_handle)
```

Question 5 (12 points)

Alice and Bob share the same bank. Before they go to dinner, they each have $200 in their bank account. When the check arrives, they execute a transaction in which Bob pays $60 to the restaurant for both their dinners; Alice then pays Bob the $30 that she owes him. Their bank allows these two payments to be made within the scope of a single transaction across the two accounts.

Suppose the banking system implements transactions using intentions lists. For the above transaction, the intentions list has the following entries:

START_Transaction T1
T1-1. Bob.balance = Bob.balance - $60
T1-2. restaurant.balance = restaurant.balance + $60
T1-3. Alice.balance = Alice.balance - $30
T1-4. Bob.balance = Bob.balance + $30
COMMIT_Transaction T1

Suppose the system crashes after step T1-3 is executed, but before step T1-4. After the system restarts and recovers, Bob sees that his account balance is $110 and Alice's account balance is $140. Clearly the execution of the transaction has been botched; the correct balances should be $170 for Bob and $170 for Alice. Identify what bug(s) in the intentions list led to this erroneous outcome. How should the bug(s) be fixed?

Question 6 (16 points)

A banking system uses write-ahead logging (WAL) based on a no-undo/redo algorithm to implement local atomic transactions. The account database is implemented as a key-value store on disk. The system supports these operations:

- get(K) obtains the value of key K
- put(K, V) sets the value of key K to V

For any transaction Tx, there will be three types of log entries:

- <Tx, start> is written to the log when Tx has just started
- <Tx, K, V> is written to the log when transaction Tx sets the value of key K to value V
- <Tx, commit> is written to the log when transaction Tx is committed, after which all unflushed trailing parts of the log are flushed

The system starts running at Time 0. From Time 0 to Time 15, three transactions T1, T2 and T3 execute interleaved as shown below:

Time	**T1**	**T2**	**T3**
0	begin (T1)		
1	X = get($\mathcal{P}$)		
2	X = X + 150		
3	put($\mathcal{P}$, X)		
4	X = get($\mathcal{Q}$)		
5	X = X − 150		
6	put($\mathcal{Q}$, X)		
7	end(T1)		
8		begin(T2)	
9		Y = get($\mathcal{S}$)	begin(T3)
10		Y = Y − 50	W = get($\mathcal{P}$)
11		put($\mathcal{S}$, Y)	Z = get($\mathcal{Q}$)
12		Y = get($\mathcal{R}$)	put($\mathcal{P}$, Z)
13		Y = Y − 200	put($\mathcal{Q}$, W)
14		put($\mathcal{R}$, Y)	
15		end(T2)	

A. Suppose accounts $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$, and $\mathcal{S}$ have initial balances of 100, 200, 300, and 400 respectively. Assuming no truncation has happened, show the contents of the write-ahead log after Time 7 but before Time 8.

B. Suppose a log truncation occurs after Time 7. The system continues to run until the server crashes after Time 15.

   (i) What would the log contents be before crash recovery begins?

   (ii) Upon crash recovery, which transaction(s) should be redone? Explain your answer in one sentence.

C. From (B), assume that the crash recovery completes successfully. What are the final values of $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$, and $\mathcal{S}$ after the recovery? (i.e. If another transaction performs get($\mathcal{A}$), get($\mathcal{B}$), get($\mathcal{C}$), get($\mathcal{D}$), what are the return values respectively?)