Problem Set 2

15-440/15-640 Distributed Systems Spring 2026

Assigned: Thursday, February 5, 2026

Due: Friday, February 13, 2026 at 11.59 pm

Submission procedure:

- Create a **.pdf** of your answers and upload to `Gradescope`. All enrolled students should have Gradescope accounts. Ask **well in advance of the deadline** to be added to Gradescope if you don't have access; it is unacceptable to ask to be added immediately before the deadline.
- 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)

Aditya, Maelle, and Hunter are writing a research paper together. The research paper manuscript is stored as multiple LaTeX files on a CMU server, and the authors access the files on their laptops. Their laptop's filesystem performs on-demand, whole-file caching of the files on the server and provides session semantics at open-close granularity. Assume that the laptop caches are large enough to hold all of the relevant files. They are trying to decide which cache consistency strategy they should use: check-on-use, callbacks, or leases.

A. For each of the following scenarios, which strategy is best? Explain briefly (in 1-2 sentences), using the considerations discussed in the lectures.

   a. The authors are writing to the same file at non-overlapping periods during the day (e.g., passing a metaphorical "write lock" out-of-band)

   b. The authors are writing to different files and occasionally check each other's progress

   c. The authors frequently need to update the same citations file

   d. Maelle and Hunter keep opening and closing `results.tex` every 5 minutes to see if Aditya has made his single, large update to the file.

B. The submission deadline is close, but Aditya is still writing. There are 10 total files, each 10 KB in size, and Aditya may be editing any of them. Assume the edits do not change the file sizes. Maelle and Hunter each check on Aditya's progress five times over the afternoon.

   a. If there is no caching on the laptops, how much total data will be sent over the network due to Hunter and Maelle checking on Aditya's progress?

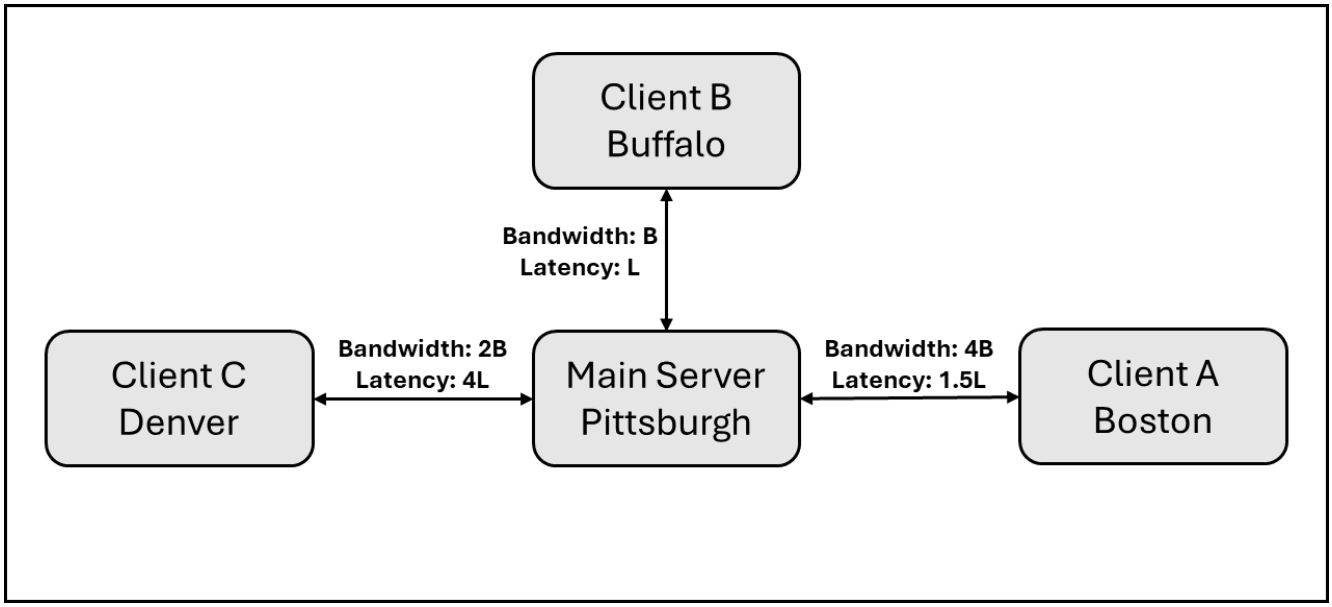
   b. Now assume the laptops use a **check-on-use** caching policy. As Aditya is hard at work, at any given time, the probability that one of the files has been updated since the last read is 20%. Assume checking the validity of a cached file involves sending a 0.1 KB message to the server, which responds with either the new file contents or a 0.1 KB message indicating the file hasn't changed. What is the expected total amount of data sent on the network due to Hunter and Maelle checking on Aditya's progress?

   c. Now assume the same scenario as (b), except the laptops now use a **callback-based** caching policy that sends a request for the new file on each callback break. There is a 20% probability that the file is changed before each check and 80% probability that it is unchanged. What is the expected total amount of data sent over the network? Assume 0.1 KB of data is sent for a callback break, and getting the newest version of the file involves sending a 0.1 KB request to the server.

   d. Now assume the laptops use file-level **leases** with a 24h lease duration. Aditya currently has the write lease and is still working. Requesting a lease involves sending 0.1KB to the server, and the response is either 0.1KB or they receive the new file. Maelle and Hunter still attempt to each check Aditya's progress five times over an afternoon. How much total data will be sent over the network for these checks?

Question 2 (20 points)

SharedStitches is an app that lets users view, share, and rate the hottest new designs and patterns for various yarn, thread, and fabric-based hobbies. Users of the app are able to browse a centralized repository for patterns based on the type of craft and category of the pattern. Users may also upload new patterns that they find or create for others to see, or update existing patterns with comments, corrections, or ratings. The current system architecture routes all requests to a single server hosted on-premise at the company's headquarters in Pittsburgh. The users, however, are scattered throughout the country; as a result, users experience varied rates of responsiveness and delay when trying to access their feed of currently hot craft ideas.

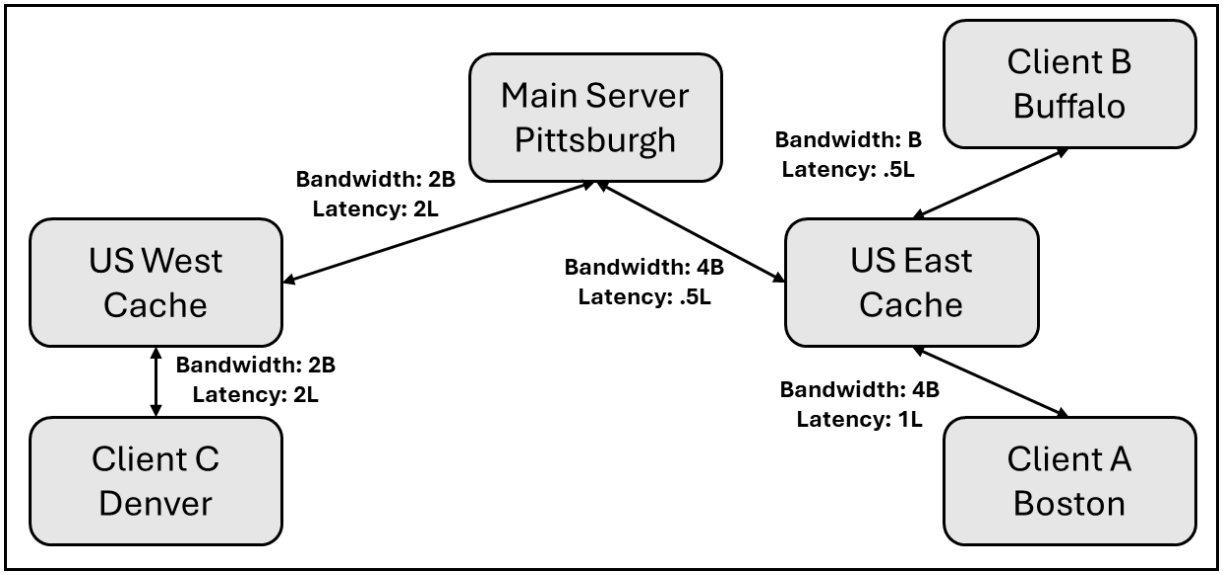


A. Suppose users A, B, and C, located in Boston, Buffalo, and Denver, each click to view a particular pattern. To request the pattern, the client-side of the application sends a request packet of size R to the server in Pittsburgh, which then responds by sending a data packet of size P containing the desired pattern back to the client. Ignoring processing and queuing delays, how long does it take between clicking until each user can view the pattern?

B. As SharedStitches grows in popularity around the world, a common user complaint is that the app feels a little slow to respond and complete requests. A member of the development team proposes caching as a possible solution, and suggests implementing a cache directly into the client-side of the app. Assume that the system implements client-side caching using a check-on-use policy, and that the time to check the local cache is negligible. To check the version at the server, the client sends a request of size C. If the version is still current at the server, it replies with a response also of size C; otherwise, it sends the latest version of the pattern of size P. How long would it take before each user can view the desired pattern assuming a cache hit and no updates at the server? Which client benefits the most from caching and why (1-2 sentences)?

C. Suppose that instead of opting for a check-on-use policy, the system implements a callback policy. Break messages are of size C and are sent to all clients immediately after the server receives an updated version of a pattern. How long does it take before a user in each city can view their requested pattern on a cache hit?

D. While client-side caching sounds promising, you point out to the rest of the team that it works well only if users view the same patterns over and over again. Doom-scrolling through the latest trending patterns is not likely to have a reasonable hit-rate. As an alternative to client-side caching, you propose establishing two proxy servers, one on either side of the country, to serve as caches between the client and the server in Pittsburgh, as shown below.



Assuming a callback-based caching policy and a cache hit, how long does it take between clicking and viewing the pattern for each user? Which clients benefit the most and least relative to the system in (A) and why (1-2 sentences)?

E. User B in Buffalo updates a pattern and clicks on upload, sending a message of size R + P to the proxy server. The proxy relays the message to the main server, which instantaneously updates the pattern once the message is fully received. Assume a callback-based policy, and that on a cache miss, the proxy servers request the pattern from the main server using the process outlined in (A). What is the shortest possible interval from when user B clicks update until user C in Denver can see the updated pattern, assuming the pattern was cached at US West? How about if the cache was cold?

Question 3 (20 points)

Greg and Bobby are collaborating on their French assignment, where they are tasked with translating as many unique fruits to French as possible. As Greg and Bobby are busy CMU students, they decide to work asynchronously, each adding translations whenever they think of one.

They write their translations to a shared `translations.txt` file, which is stored on a distributed file system that uses whole-file caching with session semantics. Each of their laptops has its own cache.

The `translations.txt` file is initially empty. In order to add their translation, they use a function called `append`. This function first runs the `grep` program twice to check whether the English or French word already exists in the file (to avoid duplicates or synonyms for the same fruit). If neither finds matches in the file, then it actually adds the new translation to the file by opening `translations.txt`, seeking to the end of the file, writing the supplied translation string to a new line, and closing the file. Recall that with session semantics, an updated file is propagated to the server on close. You may assume that the complete append function takes negligible time to execute (e.g., less than one second).

The following table shows the timeline of operations performed by Greg and Bobby on `translations.txt`.

Time	Greg	Bobby
10:30	append(“apricot = abricot”)	
11:00		append(“banana = banane”)
11:30		append(“orange = orange”)
12:00	append(“apple = pomme”)	
12:30	append(“grape = raisin”)	
1:00	append(“cherry = cerise”)	
1:30		append(“grape = raisin”)
2:00	append(“pear = poire”)	
2:30	append(“pineapple = ananas”)	
3:00		append(“plum = prune”)
3:30	append(“lemon = citron”)	
4:00		append(“raspberry = framboise”)
4:30		append(“mango = mangue”)

A. Suppose the system implements check-on-use caching, what will be the contents of Greg's copy, Bobby's copy, and the server's copy of `translations.txt` at:

    a. Time = 2:45

    b. Time = 4:45

B. Suppose the system implements faith-based caching with a TTL of 3 hours, what will be the contents of Greg's copy, Bobby's copy, and the server's copy of `translations.txt` at:

    a. Time = 2:45

    b. Time = 4:45

C. Describe one advantage and one disadvantage of each of the caching policies used in (A) and (B) above.

Question 4 (20 points)

Fintendo is developing a large-scale, open-world game with hundreds of non-player characters (NPCs). During gameplay, the frontend running on a gaming console will frequently access data for many different NPCs (for example, the NPC's position and dialogue state) that it must fetch from the backend server in order to correctly render the game. To improve performance, Fintendo engineers want to add a caching mechanism that can store data for a limited number of NPCs on the console. There is space for 6 items in the cache, which starts out empty. The team wants to understand how different cache algorithms behave, so they record sequences of NPC accesses made by the game rendering engine. For the following questions, assume that each number corresponds to the id of a unique NPC.

A. Suppose the system is presented with a sequence of NPC accesses represented by the following reference string:

   "1, 2, 3, 4, 5, 6, 3, 2, 1, 7, 5, 6".

   a. If the cache uses an FIFO replacement policy, what is the cache state (from oldest to newest) after the entire reference string is processed? What is the hit count and hit ratio of the cache on this reference string?

   b. Suppose the cache replacement policy is changed from FIFO to LRU. Repeat (a) for this replacement policy, and give the final cache state from least to most recently used.

B. In a different game scenario, a different reference sequence is generated:

   "0, 1, 2, 3, 4, 5, 0, 1, 2, 6, 7, 8, 0, 2, 1, 6, 5".

   a. If a FIFO cache replacement policy is used, what is the final cache state (from oldest to newest), hit count, and hit ratio?

   b. If LRU is used instead, what is the final cache state (from least to most recently used), hit count, and hit ratio?

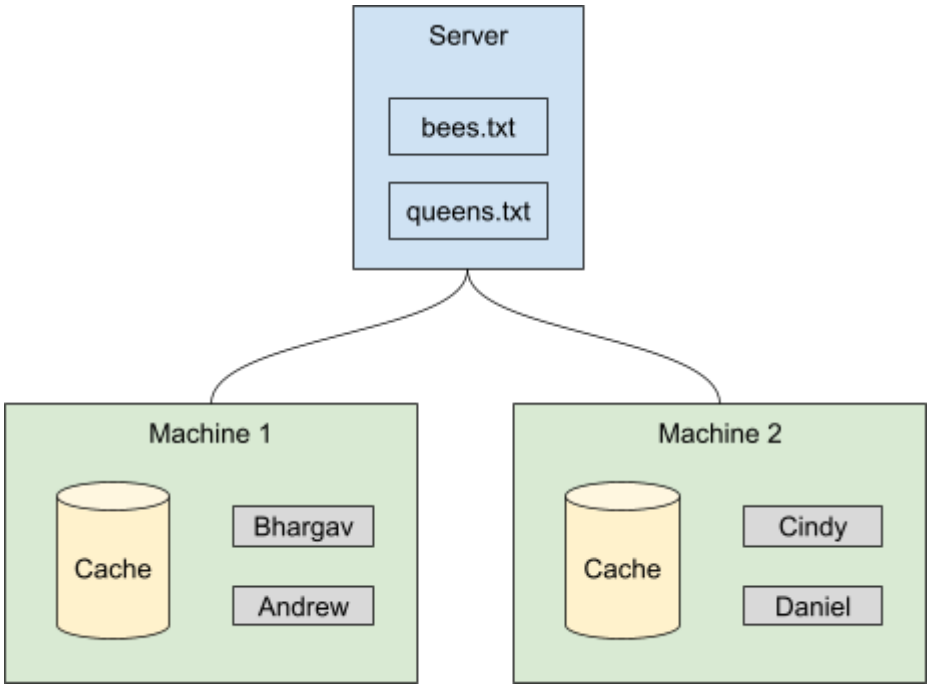
C. In which scenario does LRU outperform FIFO, and why?

D. Can there be a reference string where neither LRU nor FIFO performs well? If not, explain why. If so, give an example reference string and explain what happens.

Question 5 (20 points)

Welcome to the Global Honey Conservation (GHC) 🐝 society at CMU, where we care a great deal about honey bees and their well-being! Andrew, the head (and currently the only) apiarist, is trying to spread awareness about bees. Part of his effort includes keeping an updated count of the total number of bees across his hives in a file called “bees.txt”. He also maintains a file which tracks the count of just the queen bees, called “queens.txt”. Each of these files contains just a single line with the corresponding count value, and is stored on a central server.

Currently, a few of the other members, Bhargav, Cindy, and Daniel, are following his work, and keep track of updates made to his two files. They use a distributed file system pictured below. The system uses whole file caching with a check-on-use / open update policy, separate caches per machine, and provides open-close session semantics (e.g., like AFSv1). Recall that in such a system, any updates are propagated to the server when the modified file is closed.



The table below shows a timeline of concurrent file operations performed by the four GHC members. In reading the table, keep in mind:

- readCount(fd) is a custom read function, which seeks to the beginning of the file and reads the count value.
- incCount(fd, delta) is a custom write function, which reads the count, increments by delta, and completely replaces the value in the file.
- Assume all operations in T_i finish before T_{i+1} begins, network latency is negligible, and there are no failures of any kind (network, server, or client).
- Initially all caches are empty and both files store a count of 0.

Time	Andrew (Machine 1)	Bhargav (Machine 1)	Cindy (Machine 2)	Daniel (Machine 2)
T1	bee = open("bees.txt")	bee = open("bees.txt")	bee = open("bees.txt")	bee = open("bees.txt")
T2		readCount(bee)	readCount(bee)	
T3	incCount(bee, 1)			
T4		readCount(bee)		readCount(bee)
T5	close(bee)	close(bee)	close(bee)	
T6	queen = open("queens.txt")	bee = open("bees.txt")	bee = open("bees.txt")	
T7	incCount(queen, 2)	readCount(bee)	readCount(bee)	readCount(bee)
T8	close(queen)			
T9		close(bee)	close(bee)	queen = open("queens.txt")
T10	bee = open("bees.txt")	bee = open("bees.txt")		readCount(bee)
T11	incCount(bee, 4)			readCount(queen)
T12		readCount(bee)		close(bee)
T13			bee = open("bees.txt")	
T14	close(bee)	close(bee)		
T15			readCount(bee)	
T16			close(bee)	close(queen)

A. Given the flow of events as described in the above table, what are the values read by each of the readCount operations? Please give your answer as tuples in the following form:
([timestamp], [user name], [content read])
For example, a read by Bhargav at T27 of value 32 should be written as “(T27, Bhargav, 32)” Clearly explain your reasoning and any assumptions you make.

B. Suppose the system uses callbacks instead of check-on-open, and that the caches update immediately on receiving a callback break. Would the readCount values from part A change? If yes, list the reads that would change using the same tuple format as in part A. Clearly explain your reasoning and any assumptions you make.

C. Suppose instead the system uses the semantics from Project 2, which uses check on open, but requires that during a session (between open and close on each file descriptor), no external updates should affect the session. How would readCount values change from part A? List reads that would change using the tuple format from part A. What, if any, major changes to the caching system are needed to support this system behavior?