Problem Set 2

15-440/15-640 Distributed Systems Spring 2025

Assigned: Thursday, February 6, 2025

Due: Thursday, February 13, 2025 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 hand write 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 (15 points)

A team of CMU employees is working on a project that is spread across the Pittsburgh, Rwanda, Adelaide, and Qatar sites. Network latency between these sites is high, with unpredictable variability. Project data is kept as spreadsheet files (one per site) in a distributed file system called the Swift File System (SFS) that uses whole-file caching with session semantics. There is a single file server for SFS, and it is located in Pittsburgh. Files are cached on the desktop or laptop of each user. SFS supports a wide range of caching protocols. The choice of protocol can be configured (and reconfigured) on a per-project basis by administrative staff. The usage pattern involves opening an SFS file, briefly viewing and/or modifying it, and then closing it. Files are never kept open for long periods of time.

A. In the initial phase of the project, the team member who is visiting a site does most of the viewing and modification of that site's spreadsheet. For example, most of the file operations on the Rwanda spreadsheet are done by the team member at Rwanda; most of the file operations on the Adelaide spreadsheet are done by the team member at Adelaide; and so on. A few times a day, a team member at one site may reach out to view the state of progress at one or more of the other sites; he or she never makes a modification.

   What caching policy would you recommend for this phase of the project? Briefly explain your reasoning.

B. After the initial phase, all team members return to Pittsburgh to continue the project from their on-campus offices. In this intermediate phase of the project, team members view and modify files for all sites. In other words, there is no clear association between a team member's file accesses and the specific file for a site. The usage pattern remains the same as before: opening an SFS file, briefly viewing and/or modifying it, and then closing it. As before, SFS files are never kept open for long periods of time.

   What caching policy would you recommend for this intermediate phase of the project? Briefly explain your reasoning.

C. In the final phase of the project, the team members invite other key stakeholders at all sites to view the state of their work. These stakeholders mostly view the spreadsheet files; only very rarely do they modify them by adding a comment or question. It is important that their modifications be immediately visible to everyone else, across all sites. The per-file usage pattern (i.e., a file not remaining open for long) remains unchanged.

   What caching policy would you recommend for this final phase of the project? Briefly explain your reasoning.

Question 2 (20 points)

Helen is a boba entrepreneur. She tirelessly creates new boba recipes and updates existing recipes for her chain of boba shops. The stores are eager to use her latest recipes, as soon as they become available. All recipes are stored as files on a server in California, one recipe per file. Helen's boba shops nationwide are clients in this distributed file system, and cache recipes as specified in the

questions below. You may assume that caches are infinitely large, that each recipe is 1KB in size, and that whole-file caching is used. You may also assume that the Internet is perfectly reliable, with no packet loss, damage or reordering.

A. Suppose caching is done locally at each shop, and a callback-based caching policy is used. In Pennsylvania, Helen's boba shops are located in Pittsburgh, Philadelphia, and Harrisburg. The end-to-end network performance from the server to these three shops is as follows:

   - Pittsburgh: B bps bandwidth and 100 ms one-way latency
   - Philadelphia: 2B bps bandwidth and 100 ms one-way latency
   - Harrisburg: 0.1B bps bandwidth and 100 ms one-way latency

   When a new recipe is created, or an old one is modified, how long does the callback break take? Clearly state any assumptions that you make. After a new recipe is released at the server, how long is it before that recipe is available at all the Pennsylvania stores? Give a brief explanation of your calculation, and illustrate it with a timeline.

   (Hint: remember that a callback break only invalidates a cache entry. New contents are not pushed. The fetching of new contents occurs on a later client-initiated operation.)

B. In addition to caching locally, suppose the three shops in Pennsylvania also share a caching proxy in Chicago. In other words, there are two levels of caching. Both levels use a callback-based caching policy. The network performance is as follows:

   - server to Chicago proxy: 100*B bps bandwidth and 40 ms one-way latency
   - Chicago proxy to Pittsburgh: B bps bandwidth and 40 ms one-way latency
   - Chicago proxy to Philadelphia: 2B bps bandwidth and 40 ms one-way latency
   - Chicago proxy to Harrisburg: 0.1 B bps bandwidth and 40 ms one-way latency

   When a recipe is updated at the server, what is the total delay before that recipe is available at all Pennsylvania shops?

C. Helen decides to let each boba shop create its own city-specific recipes and share them with her nationwide set of shops. Regardless of where they were created, all recipes are stored on the server in California. The caching policy is changed from callback-based to check-on-use. The RPC to check if a file is up to date involves a 100-byte request packet and a 10-byte reply packet.

   Suppose no caching proxy is used, and network performance is that described for (A). A recipe is updated by the Philadelphia shop. From the moment the file is closed in Philadelphia, what is the minimum delay before it is visible to an open at the Pittsburgh shop? Explain your work, and show your reasoning using a timeline.

D. Suppose the Chicago caching proxy, with network performance as shown in (B), is used. You may assume that check-on-use is used at both levels of caching. A brand new recipe appears on the server from Harrisburg. Sometime later, the Pittsburgh shop opens this new recipe for the first time. For the best case and worst case, what is the minimum delay before the open completes? Explain your work, and show your reasoning using a timeline.

Question 3 (20 points)

A team of observers is logging the entry of people into a large room with many doors. The fire marshall strictly limits the number of people that are allowed in the room. This limit should not be exceeded. The log is stored as a single file `counter.txt` in a distributed file system. The file is initially empty. As each person enters the room, the timestamp, current headcount (including this person), and the door by which the person entered are recorded: e.g., the entry `"293, 8, G"` would mean *"at timestamp 293, the 8th person to enter the room came in through door G"*.

Each observer stands at a different door with their laptop in hand. When a person enters through that door, the observer opens `counter.txt`, reads the latest headcount in the file, adds a new line `"<timestamp, previous_headcount+1 door name\n>"`, and then closes the file. You may assume that this entire process, including update propagation to the server, happens instantaneously.

Suppose the distributed file system uses whole-file caching with session semantics. In case of write sharing, the last close wins. Each laptop has its own cache. There is no shared caching proxy.

Suppose the following pattern of people entering the room is observed:
(an "X" means entrance at the timestamp corresponding to that row, via the door corresponding to that column)

Time (seconds)	Door A	Door B	Door C	Door D	Door E
1	X				
2		X			
3	X				
4			X		
5				X	
6			X		
7			X		
8					X
9		X			
10	X				

A. Suppose check-on-use is used as the caching policy.

   i. What are the contents of observer A's cached `counter.txt` right after t = 3?
   ii. What are the contents of observer C's cached `counter.txt` right after t = 7?

B. Suppose faith-based caching with a TTL of 3.5 seconds is used instead of check-on-use.

i. What are the contents of observer A's cached `counter.txt` right after t = 3?
ii. What are the contents of observer C's cached `counter.txt` right after t = 7?

C. Describe one advantage of each of the caching policies used in (A) and (B) above. For this application, which method would you recommend and why?

Question 4 (20 points)

Ruiqi's board game has become a smash hit. One of her most loyal fans, Michael, suggests a cool new feature — letting players revisit past game states to learn from their mistakes. Sunny offers to add a caching mechanism to speed up the revisiting of old game states. Each entry in the cache is the entire game state at some point in the past. There are 8 cache entries. For the following questions, assume that each number corresponds to a unique game state.

A. Suppose the cache uses an LRU replacement policy. It is presented with the following reference stream:

"71 72 73 74 75 76 77 78 71 79 72 73 74 75 76 77 78".

What is the cache state after the entire reference stream is processed? What is the hit ratio of the cache on this reference stream?

B. Suppose the cache replacement policy is changed from LRU to FIFO. Repeat (A) for this replacement policy.

C. Suppose a different reference stream is used:

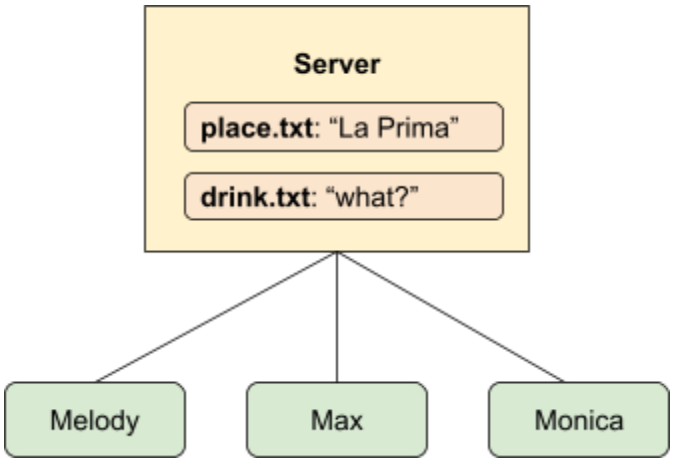
"70, 71, 72, 73, 74, 75, 76, 77, 70, 71, 72, 73, 78, 79, 80, 70, 72, 71, 77, 73"

a. If an LRU cache replacement policy is used, what is the final cache state and hit ratio?

b. If FIFO is used instead of LRU, what is the final cache state and hit ratio?

D. What do the results suggest about the effectiveness of different cache policies in relation to temporal locality? Can there be a reference stream for which neither LRU nor FIFO performs well? If no such reference stream can exist, explain why. If such a reference stream is possible, give an example with explanation.

Question 5 (25 points)



Melody, Max, and Monica are on three different clients connected to a file server storing files `place.txt` and `drink.txt`. The initial contents of the files are as shown in the figure above. Each client has its own local cache, which starts off empty and is infinitely large. The cache uses whole-file caching with open-close session semantics and check-on-use. Here are some additional clarifying details that may be relevant:

- Recall that check-on-open with session semantics implies store-after-close. For a set of concurrent `open()` operations on a file at a client, only the first involves a check (and possible fetch). No further checks are done for that file until the set becomes empty (i.e., a `close()` has been issued for every one of the concurrent `open()` operations). While a file is open at a client, no changes on the server are visible; the only visible changes during this period arise from local `write()` operations to the file. The store of a modified cache copy is only performed after the last `close()` of the set of concurrent `open()` operations by the same client.
- `read()` and `write()` operations always start at the current file pointer, that is internally maintained separately for each `open()` (i.e., it is per-file-descriptor). The file pointer is set to zero at `open()`.
- `read(fd, n)` reads `n` bytes from the current read pointer for `fd`, and advances the file pointer by `n`; the bytes read are returned as the value of the call. An end-of-file exception is raised if an attempt is made to read more bytes than remain to be read.
- All operations in T_i finish before T_{i+1}.
- Network latency is negligible and can be safely ignored.
- There are no failures of any kind (network, server or client).
- The bytes written are exactly as shown in the `write()` calls below; e.g. no extra null terminator is added for strings.

Time (T)	Melody	Max	Monica
1	fd1 = open(“place.txt”)		fd1 = open(“drink.txt”)
2		fd1 = open(“place.txt”)	write(fd1, “Oolong milk tea”)
3	write(fd1, “Mango Mango”)		close(fd1)
4	**read(fd1, 11)**	**read(fd1, 3)**	fd1 = open(“drink.txt”)
5	close(fd1)	write(fd1, “Wushiland”)	write(fd1, “Grass jelly”)
6		close(fd1)	
7	fd2 = open(“drink.txt”)		fd2 = open(“place.txt”)
8	write(fd2, “Smoothie”)		**read(fd2, 12)**
9	close(fd2)		close(fd2)
10		fd2 = open(“drink.txt”)	fd3 = open(“drink.txt”)
11		**read(fd2, 6)**	**read(fd3, 5)**
12		close(fd2)	close(fd3)
13			close(fd1)
14		fd2 = open(“drink.txt”)	
15		**read(fd2, 11)**	
16		close(fd2)	

A. For the `read()` operations shown in the table, indicate its result. Please give your answer as tuples in the following form:

(*[client name]*, *[timestamp]*, *[content read]*)

If the `read()` operation raises an end-of-file error, put “EOF error” as the content read in the tuple.

B. List the contents of the cache at each of the clients just before each of the time indicated below. Please include which files are present in each cache and their contents.

T	Name	Cache Content
2	Melody	
	Max	
	Monica	
11	Melody	
	Max	
	Monica	
17	Melody	
	Max	
	Monica	

C. Suppose callback-based caching is used instead of check-on-use. Suppose all clients start out with warm caches (i.e., they have both files in their cache before T = 1). List the contents of the cache at each of the clients just before each of the times indicated below. Please include which files are present in each cache and their contents.

T	Name	Cache Content
4	Melody	
	Max	
	Monica	
11	Melody	
	Max	
	Monica	