Problem Set 2

15-440/15-640 Distributed Systems Spring 2024

Assigned: Thursday, February 8, 2024

Due: Thursday, February 15, 2024 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)

Aditya and Ryan are collaborating on a project. They write code in a shared directory `ourproject` of a Linux server, in subdirectories `ourproject/aditya` and `ourproject/ryan` respectively. They are traveling to different destinations for spring break, but want to continue collaborating on their project. They use a tool called `easysync` to efficiently synchronize `ourproject` on the server with a local directory on each of their laptops. `easysync` is run manually by them, whenever they would like to synchronize state.

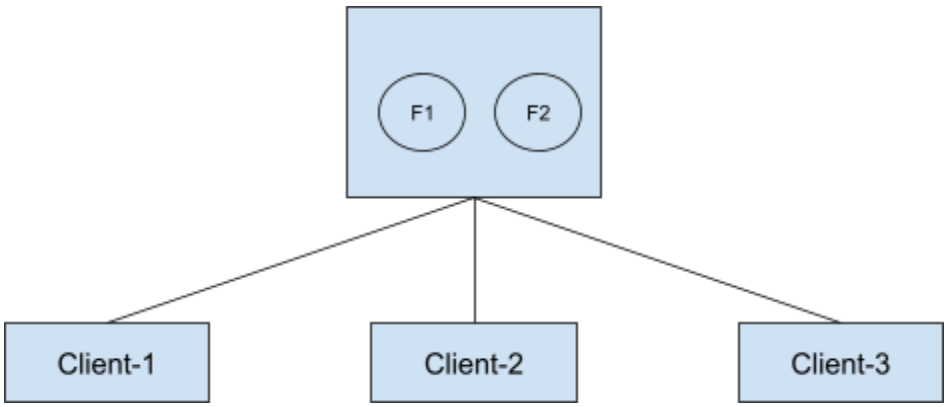
`easysync` works by computing the SHA256 hash of each file in `ourproject` at a laptop, transmitting the hashes to the server, and comparing them to the hashes of files with the same name at the server. This comparison shows which files have changed. `easysync` then transfers the newer version of the file to replace the older version.

Because of a mistake made by Aditya in setting file permissions, random people on the Internet are able to copy files into `ourproject/aditya` on the Linux server. Suppose 42 videos, totaling 10GB in size, are copied into that directory by spammers. Apart from these spam files, the directories `ourproject/aditya` and `ourproject/ryan` each have 3 project-related files. The 3 project-related files total 1 GB in size in each directory. Ryan forgets to do `easysync` before leaving on spring break, and therefore his laptop has an empty `ourproject` directory. His first `easysync` happens while traveling.

a) Suppose Ryan is using his Verizon cellular data plan for Internet connectivity. Above his monthly 4GB quota, he must pay $10/GB. How much will Ryan pay for the first `easysync` on his trip?

b) Ryan is alerted by Verizon of the sudden increase in his cell phone bill. He decides to use `gdSync` rather than `easysync`. The difference is that `gdSync` ignores a new file until the first attempt is made to open it. You may assume that this detection happens “by magic.” You may also assume that the `open()` blocks until the copy of the file bits has completed. After that first open, `gdSync` works identically to `easysync` on that file. Since Ryan’s first `easysync` while traveling, another 42 spam videos totaling 10 GB have been added to `ourproject/aditya` on the server. How much will it cost Ryan for his second easysync, if it happens after he adds another file in `ourproject/ryan` with a size of 100MB? Clearly state and explain any assumptions that you make.

Question 2 (20 points)



Three clients (different computers) are connected to a file server storing files F1 and F2. Each client has its own cache which starts cold, and is infinitely large. The cache implementation is whole-file caching with open-close session semantics, maintained by check-on-open and store-after-close. You may assume the following:

- F1 and F2 are initially empty (i.e., zero bytes long).
- Recall that check-on-open and store-after-close work as follows. 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 to the server is only performed after the last `close()` of the set of concurrent `open()` operations.
- `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 `write()` calls below; e.g., no extra null terminator is added for strings.

Time (T)	Client-1	Client-2	Client-3
01	`fd1 = open(F1)`	`fd1 = open(F1)`	`fd1 = open(F2)`
02	`write(fd1, "440")`	`fd2 = open(F2)`	
03	`fd2 = open(F1)`	`read(fd1, 1)`	`write(fd1, "hi")`
04	`read(fd2, 3)`		
05		`close(fd1)`	`close(fd1)`
06	`write(fd1, "/")`	`read(fd2, 1)`	
07	`close(fd2)`		
08	`close(fd1)`	`close(fd2)`	
09			`fd1 = open(F1)`
10		`fd1 = open(F2)`	`write(fd1, "640")`
11	`fd2 = open(F2)`		`close(fd1)`
12	`fd1 = open(F1)`	`read(fd1, 2)`	
13	`read(fd2, 1)`		
14	`read(fd1, 1)`	`close(fd1)`	
15	`close(fd1)`		
16	`close(fd2)`		

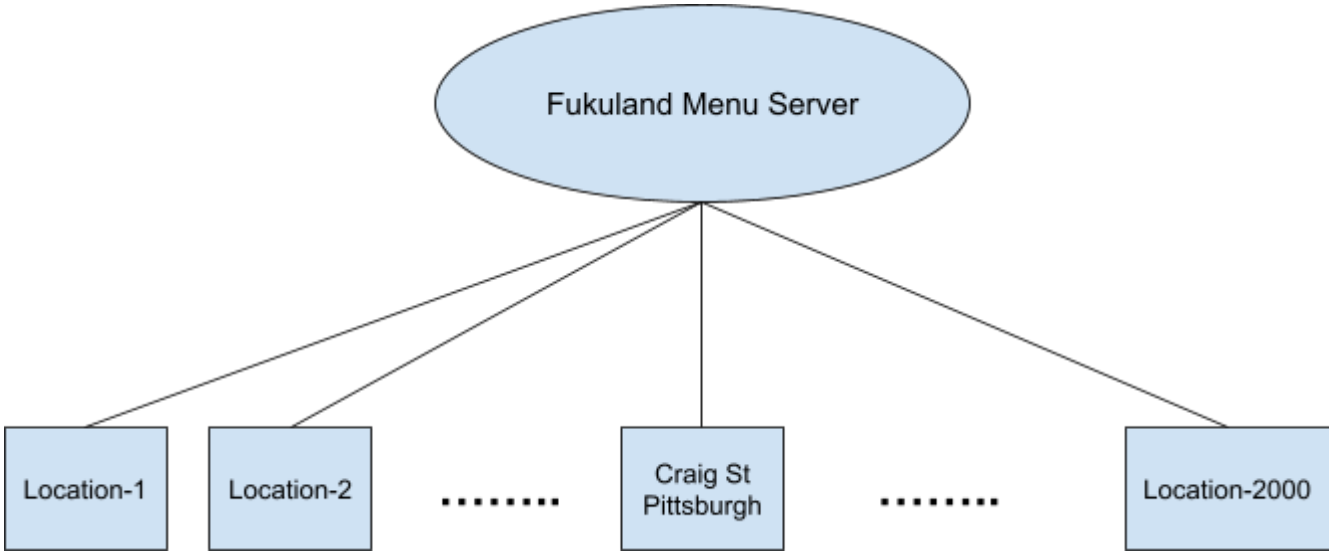
A. List the contents of the cache at each client just before each of the times indicated below (i.e., before the operation if any in that time slot is performed). Indicate which files have cache copies present, and show the contents of each copy.

   a. T = 6

   b. T = 17

B. For each `read()` operation shown in the table, indicate the result of that `read()` operation. Give your answer as a tuple of the name of the client, the timestamp, and the result of the `read()` operation. E.g. (Client-1, T25, “xxx”).

Question 3 (25 points)



A new boba shop called Fukuland has opened on Craig Street. This is a specialty boba shop that has a constantly-evolving set of 1000 unique boba recipes on its menu. Each recipe is 10 KB of information. Fukuland already has 2000 locations across America. Each location caches the current set of 1000 recipes from the central Fukuland Menu Server. Once a particular recipe receives four one-star reviews, it is deemed awful and deleted from the menu forever. It is immediately replaced by a new recipe. For high customer satisfaction, it is important that each location reflects the most up-to-date version of the menu. Customers often text their friends in other parts of the country, and are unhappy when a new boba recipe that their friends love isn't available promptly for them to try out.

Answer the following questions, assuming that each of the 2000 US-wide locations receives an average of 10,000 orders a day. The relevant recipe is needed to service each order. Clearly state and explain any additional assumptions you make.

A. Assuming no caching of recipes, calculate how many bytes are transferred per day from the server to a typical location.

B. Suppose an 8 MB cache is added at each location. Assuming that a customer's order is a uniformly random selection from the menu. Suppose there are no menu changes for an extended period of time (i.e., no cache invalidations). Calculate how many bytes are transferred per day from the server to a typical location.

C. Based on positive experience with caching, the cache size at each location is enlarged to be 100 MB. A check-on-use mechanism is used for cache consistency across locations. A period of rapid menu change occurs, and results in a 10% chance of a check indicating that the cache entry is not valid. Calculate how many bytes are transferred per day from the server to a typical location.

D. Suppose a callback-based mechanism is used to maintain cache consistency across locations (assume we still have a 100MB cache). As recipes are replaced by new ones, all locations that have that recipe cached are notified of the change. If a menu item has a 40% chance of being deleted and replaced on a given day, calculate how many bytes are transferred per day from the server to a typical location.

E. Qualitatively, what do your answers to C and D suggest about the conditions under which check-on-use is more efficient than callbacks, and vice versa?

Question 4 (25 points)

We are all familiar with how Google Docs work - a live, collaborative platform for editing documents. Let's take a look at what happens if Google Docs uses leases. For simplicity, we will assume that all leases are write leases (i.e., give exclusive access to the entire file). When multiple visitors try to grab a lease at the same time, FIFO queuing is used to serialize their access.

David is planning a massive meet-and-greet for Erin. He creates a sign-up sheet where visitors can select one of the 100 available time slots by writing their name adjacent to their preferred slot. The use of leases avoids concurrency issues across visitors. People who sign up early get a wide range of time slots to choose from; later visitors have to settle for what's left.

A. Assume that network latency is 100ms each way, that it takes 10ms for the server to grant a lease, and that the lease request and reply messages are of negligible size. What is the shortest time that a visitor has to wait to be able to sign up? Clearly state any additional assumptions that you make.

B. Suppose there are 30 possible visitors who all rush to obtain a lease immediately. A lease is granted for 3 minutes, after which it is revoked. What is the longest time that a visitor has to wait to be able to sign up? Clearly state any additional assumptions that you make.

C. John managed to sign up, but forgot to note his meeting time in his Google calendar. He now wants to revisit the sheet, just to confirm the time (but not modify it). What are the best case and worst case times for him to do this time confirmation?

D. For people like John, who just wish to view the sign-up sheet but not modify it, David wants to make checking as fast and painless as possible. He changes the FIFO queue to prioritize all readers. What possible problem do you see arising as a result of this change?

E. How would you fix the problem that arises in part D?

Question 5 (15 points)

QuantumQuery is a state-of-the-art digital assistant that processes voice queries using local computations on data cached from the cloud. It is implemented as three separate processes, each with a unique data access pattern. The cache can hold 8 data blocks, managed by an LRU eviction policy. Insertion and eviction are done on a per-block basis (i.e., there is no read-ahead or cache line concept). There are a total of 100,000 data blocks in the cloud. The unique data access patterns of the processes are as follows (numbers shown are data block IDs):

- Process-1 : performs a long sequential scan:
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ….
- Process-2 : accesses blocks in a pseudo-random sequence with no immediate repeats:
  14, 23, 32, 45, 56, 67, 78, 89, 734, 121, 4, 1098, ….
- Process 3 : repeatedly accesses the same set of blocks in the same sequence:
  33,34,35,36,37,33,34,35,36,37,33,34,35,36,37....

A. What is the expected steady-state hit rate if only Process 1 is running?

B. If only Process 2 is running, what is the hit rate at the end of the sequence shown above?

C. What is the expected steady-state hit rate if only Process 3 is running?

D. When all three processes are run concurrently, the following pattern is observed on one particular execution:
33,34,33,34,34,34,35,36,37,38,47,56,47,52,56,81,81,33,33,34,35,36,36,36

   a. What is the hit rate for this execution?

   b. What are the cache contents at the end of the execution?