Microservice Call Graph Caching

Xiuqi Lu

2024.4.4

1 Reading

1.1 Read the article especially Sec 4, Sec 5 and Appendix A

1.2 For the example in Figure 4, draw the execution sequence (similar to Figure 3) to show how MuCache protocol solves the challenge of the "diamond" pattern. What if the write and read operations come from independent processes in S1? Draw all possible execution sequences

MuCache successfully solves the challenge of the "diamond" pattern as shown in figure 1. In such applications naive caching could lead to executions that cannot be observed without caches. However, MuCache does not retrieve the return value from the cache to preserve correctness at step (5') and (7').

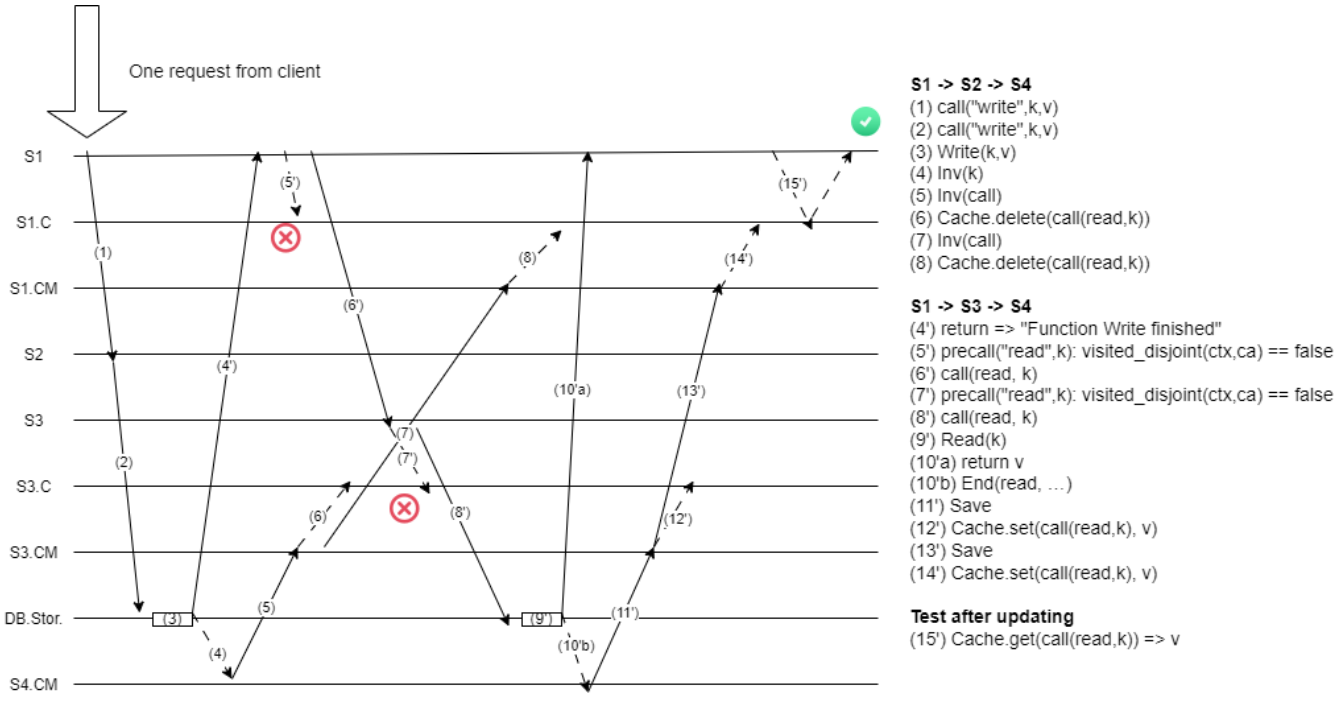

Fig. 1. Solution to the example in Figure 4

The execution of a single process in the original problem is linear, but in the new problem two independent processes perform the write and read functions. This means that the order in which the different processes execute their progress corresponds to different *ctx.visited*, which determines whether data can be fetched from the cache properly. According to the question, it seems like there should be more than two possible execution sequences, but based on my understanding of the workflow, there are only two possible orders of execution. The first one shown as figure 2 depicts when the read-process executes before the write-process, which causes *ctx.visited* to be empty, so S1 can fetch the cached value directly from its cache.

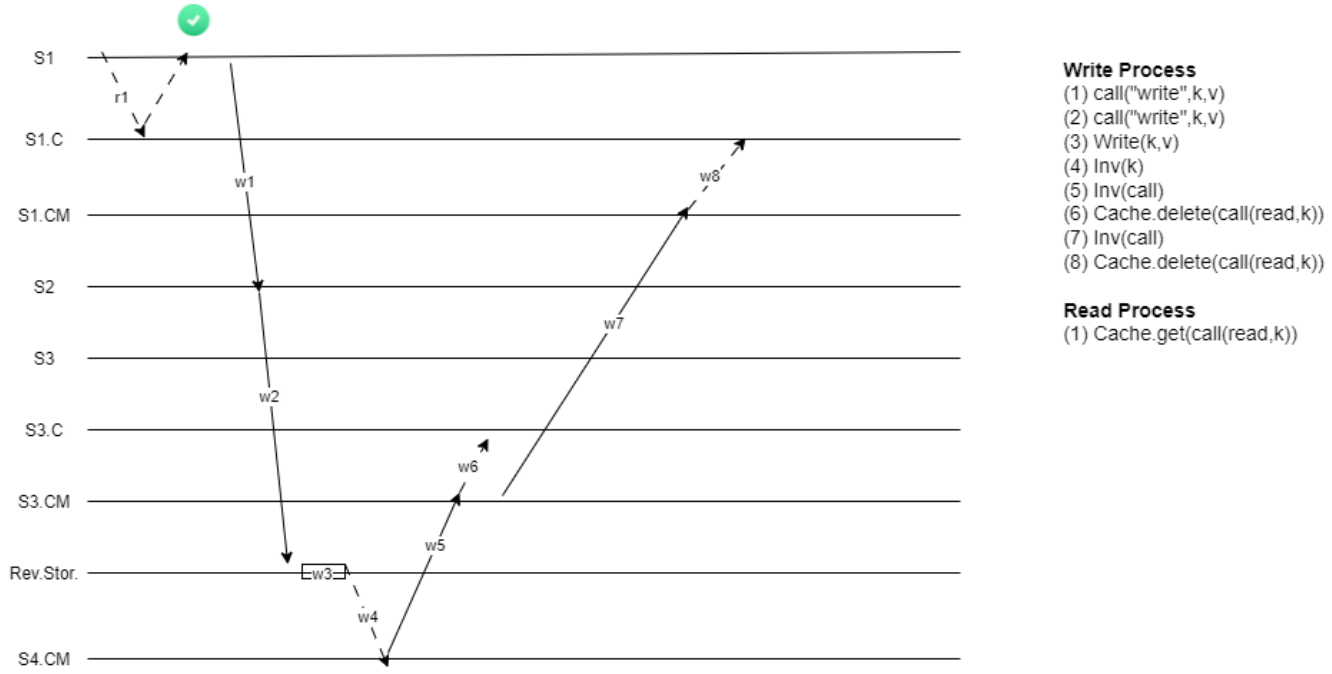

Fig. 2. Read first

The second one shown as figure 3 depicts when the write-process executes before the read-process. Here it may be refined into many cases, e.g., call(write,k,v) proceeds to S2 before the function call(read,k) is called by S1. However, I think the *ctx.visited* of the initial request to service S1 will not updated by the sub-request until S4 has executed the function write(k,v) and returned (this is a backtracking-update process) according to the paper: ***Whenever a subrequest ca returns, the parent request adds all the visited services of the subrequest (ca.visited) to its own visited services (ctx.visited).*** In the meantime, S1.Cache's corresponding *ca.visited* includes services S3 and S4. S3.Cache's corresponding *ca.visited* includes service S4 according to the paper: ***When saving a cache entry for call ca, the cache manager also stores the services, S' , that were visited during the processing of ca.***
In summary, the following three conclusions are reached:

1. If the read operation of S1 occurs before the end of the write operation of S4, then both will read the data in the cache directly.

2. If the read operation of S1 occurs after the write operation of S4 (which also means that the read operation of S3 is later than the write operation), then neither will be able to read the cached data (consistent with the result of cache miss), and will fetch the data step by step from S4's database.

3. If the read operation of S1 occurs after the invalidation, the result will be the same with figure 3.

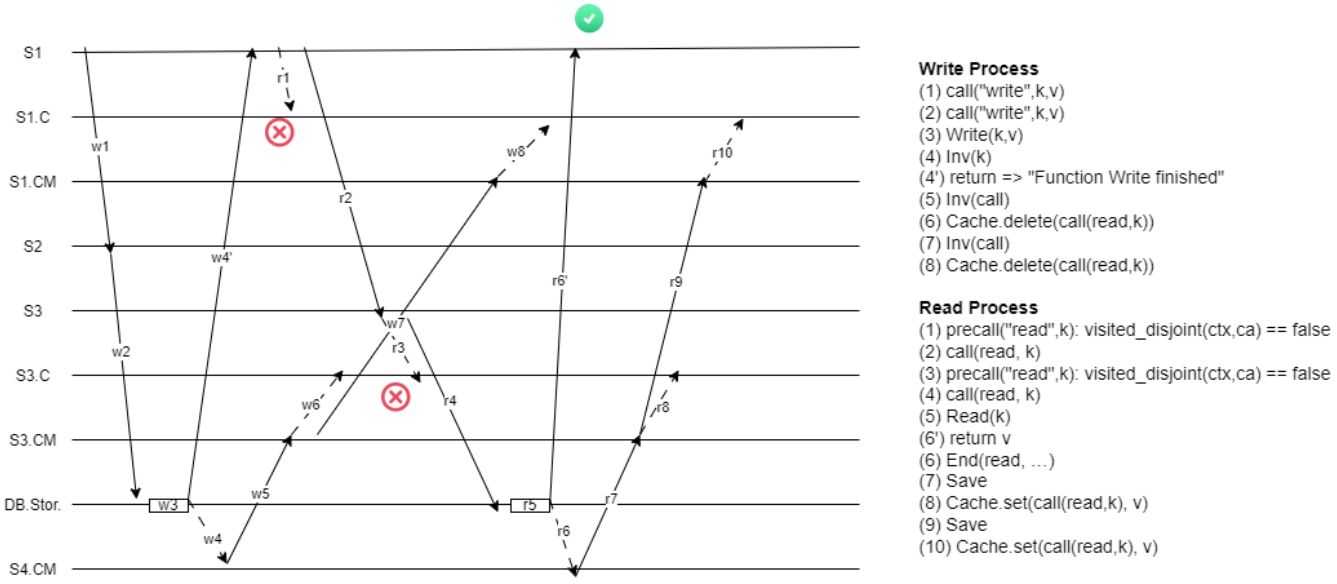

Fig. 3. Write first

Doubts in response to question 1.2:

1. Are the visited services updated recursively with the function? In the example of diamond pattern, do the visited services of the request of S1, S2 need to wait for the end of the sub-function before they are updated.

2. When saving a cache entry for call ca, where will the cache manager save these visited services?

3. What is the exact meaning of ca in the function ***visited_disjoint(ctx, ca)***? Do requests performed by different processes share the same ctx when two processes come from the same request from the client? (To deal with Q1.2, I assume that the client sends only one request to S1 and then the two process handles each subrequest.)

1.3 Choose a microservice and provide a few example function calls to simulate which data could be cached, and simulate a cache hit and a cache miss. (You may choose any cache size and eviction policy you like)

I choose movie microservice to simulate. Based on the source code of the mucache protocol, I draw the call graph between functions shown as figure 4.

Take the example of getting movie_review, the process of calling ReadMovieReviews function will trigger the ro_read_reviews function on the reviewstorage service, RO type function into Invoke will first check whether the Cache hit or not. The key-value pairs that are cached in this process consist of review_ids and reviews. If the cache is hit, the corresponding reviews are returned directly; if the cache is miss, then the Http request message is constructed and the performRequest function is called to initiate a query to the DB. Also, We will save the result to cache if we are in upperbound baseline.

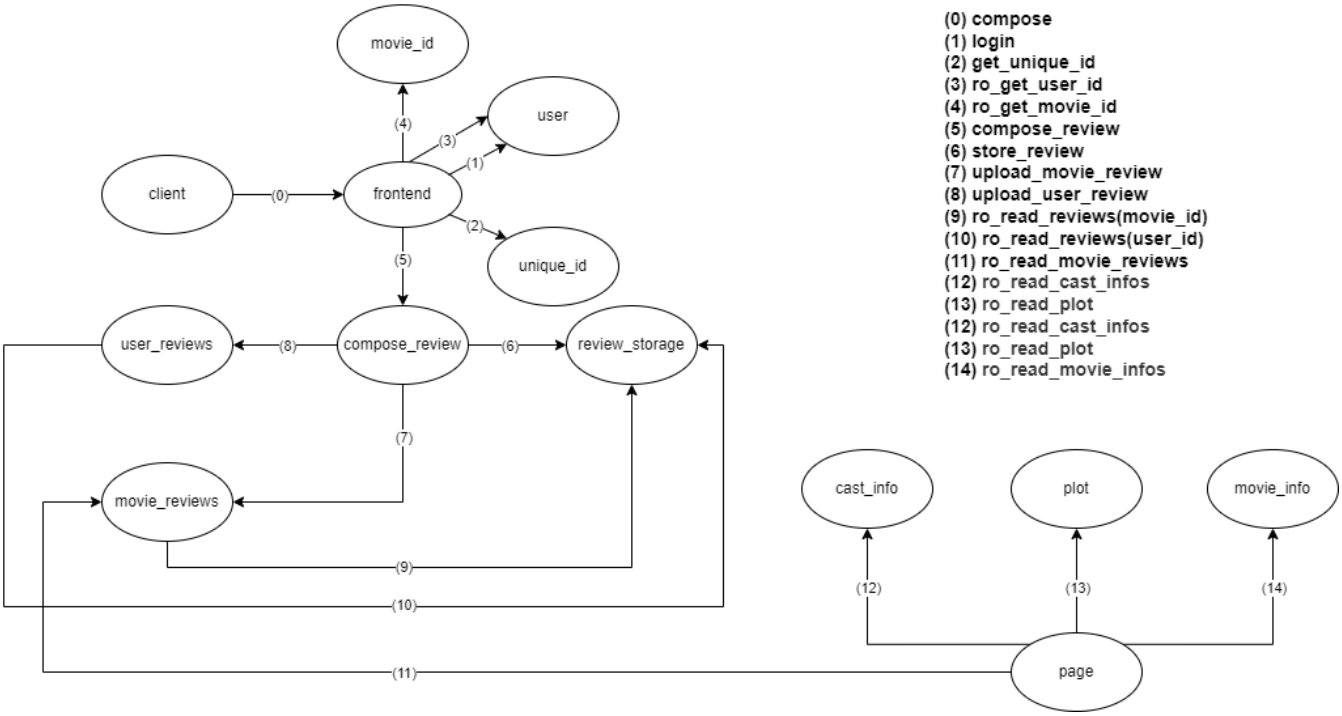

Fig. 4. Movie microservice

2 Coding

2.1 Summarize the key components to apply the MuCache protocol, and implement an example to show that the datastore is linearizable. Note: We are not asking you to build and run the open-source version of MuCache (you may use it as a reference). Please implement your own version of MuCache protocol (just a prototype to show the key ideas). You may remove any component you think is not critical in the design, and use simple functions to simulate the behavior of each service/thread in your code.

Since the code is quite complex, please view "LuCache" in the attachment, which contains the code and related instructions. The LuCache implementation comprises roughly 600 LoC of Python, including the wrappers that intercept invocations and state accesses, and the cache manager that makes invalidation and saving decisions.

3 Conclusion

3.1 What is the most challenging part of your assigned tasks?

The most challenging part is to understand the details of the workflow of MuCache protocol. In the reading part, the problem really confused me with details of execution sequences. While in the coding part, it is important to make trade-offs, focus more on the core components of MuCache Protocol and dismiss some unimportant details.