Distributed Operating Systems Labs Report

Wasim Ghanayma

Computer Engineering Department

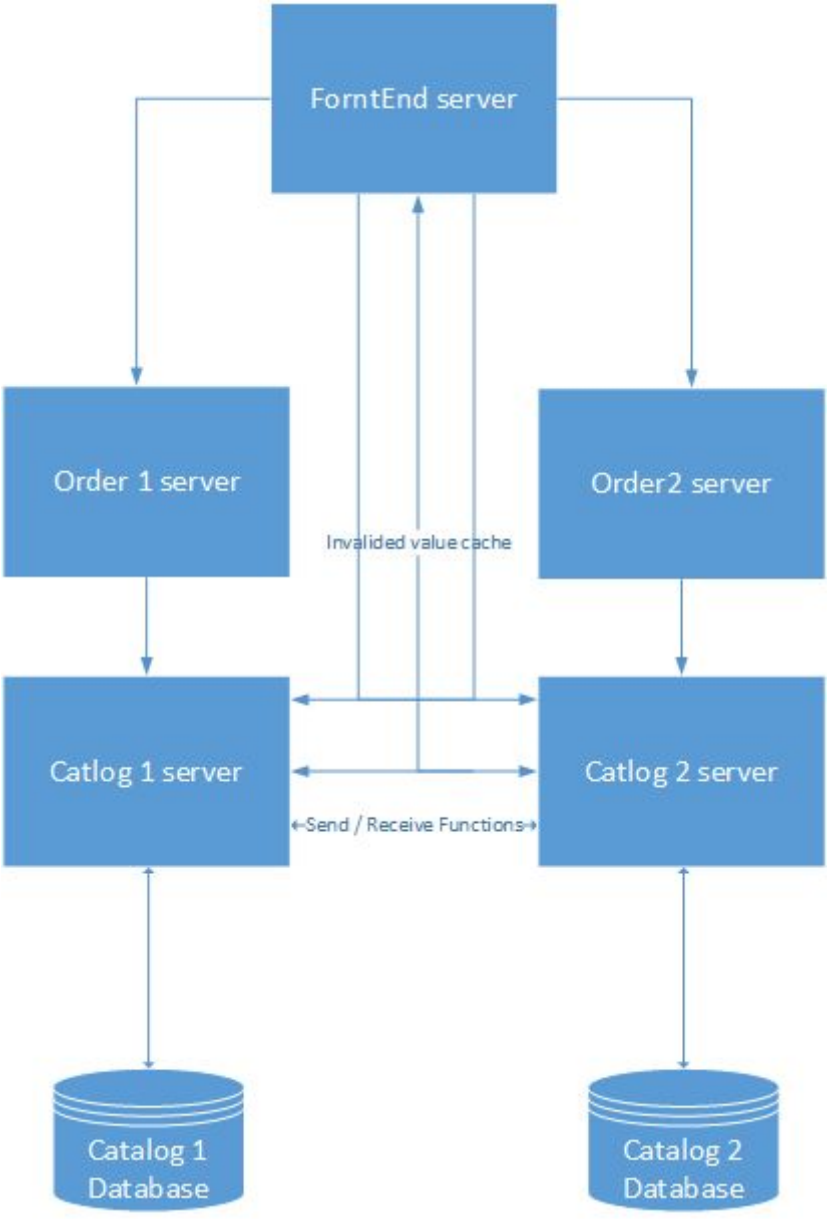
An-Najah National University

May 18, 2020

Overview :

In lab 1, we designed and implemented a layered web architecture that supported communication between different entities using HTTP REST API calls. . The lab 1 system mimics an online book store where a client can browse the books in the store and can buy a book .

In lab 2, we have improved the performance and reliability of lab 2 book store. Performance has been improved by implementing replication, load balancing, and caching .



Replication & Load Balancing

Two replicas of each catalog and order servers have been created- Catalog 1 & 2, and Order 1 & 2. Code and databases of the servers were replicated in the process. Every replica has it's own SQLite database. These replicas have been created in order to improve the performance of our system. In order to balance the incoming requests-load fairly among

the replicas, we have implemented a round-robin load balancing technique in the front end server.
For example, if the first incoming query request is forwarded to Catalog 1 replica then the second incoming query will be forwarded to Catalog 2. Order requests get forwarded to the order replicas.Here we assume that the front end will never crash and it will always stay online .

Caching

In order to improve the response time of the front end server to lookup query requests, we have implemented simple in-memory caching with limit size . The Dictionary tool of Python has been used to implement caching. Whenever a lookup request comes in, the front end server looks for the data in cache. If data is not found in cache, the request gets forwarded to one of the catalog replicas. The response received from the catalog server is then cached for quickly serving subsequent lookup requests. Cache data can get outdated if the stock or price gets updated. Invalidation request removes data for a specific item from the cache. Buy transaction, new stock addition, or price update operations cannot go through before invalidating cache. Backend servers make a REST API call to the front end server to invalidate a cache entry and wait for an acknowledgement message before changing the price or stock value

- **invalidate cache(item number)**: This interface is provided by the front end server. Back end servers- catalog and order, use this interface to invalidate front end server's cache entry for a specific book. Cache invalidation operation is performed right before changing the stock or price of a specific book. Specifically, buy operation and new stock addition cause the stock value of a specific book to be altered and so cache invalidation is performed right before these operations are performed. This interface sends an acknowledgement to the back end server stating that the cache has been invalidated, only after which any stock/price altering operation can be carried out by the back end server.

Average Response Times

Experiment	Search	Lookup	Buy
Without Caching	0.0106201171875	0.011316394805908203	0.06667327880859375

Caching Size /Experiment	Lookup	Buy
1	0.008995542526245117	0.07490797996520997

2	0.008233489990234375	0.0741870379447937
3	0.0067631244659423825	0.05906756401062012
4	0.006723995208740234	0.05983109951019287
5	0.005152621269226074	0.0851466178894043
6	0.004201858043670654	0.05425120353698731
7	0.003785469532012939 4	0.07499626398086548
100	0.003589234352111816 4	0.09990184783935546
1000	0.003966214656829834	0.09746306896209717

1. How much does caching help?
   It helped in the searches but in the buy it did not help.
2. What are the overhead of cache consistency operations?
   Stock/Buy require , so you will need to change the data within the cache and sql, which is expensive.
3. What is the latency of a subsequent request that sees a cache miss?

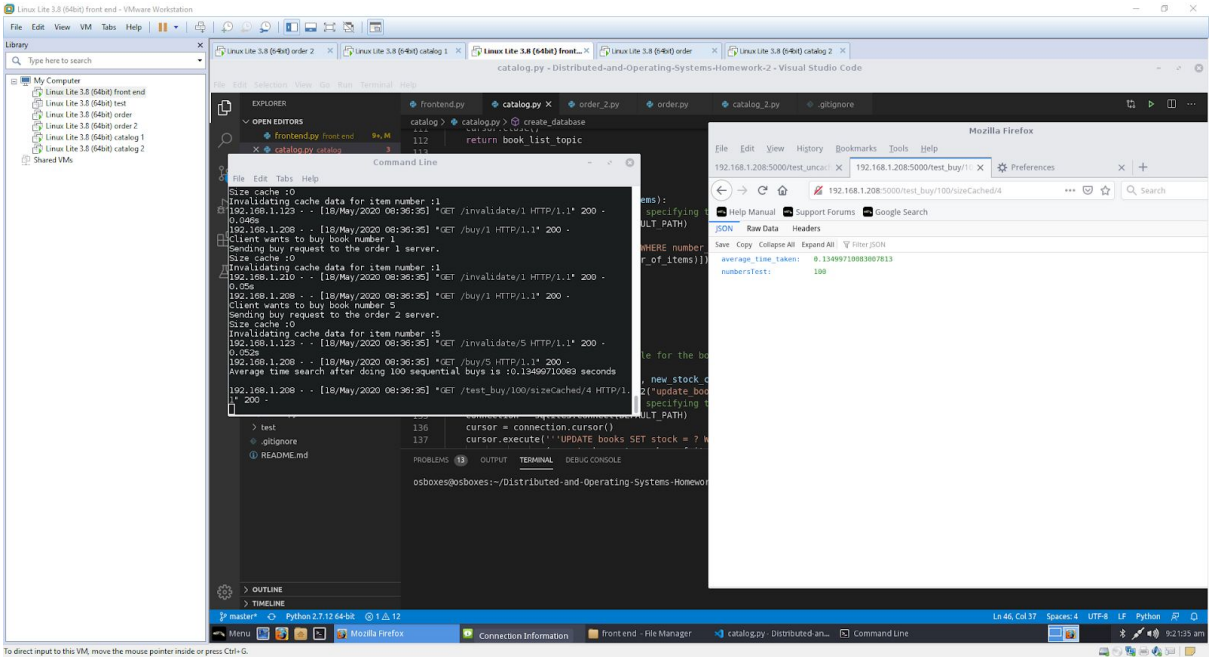
   If the response data for lookup requests is found in the front end server cache, it is immediately sent over to the client. But if the data is not present in cache, the front end server forwards the request to the back end server and then the response received is forwarded to the client. The second case is definitely time consuming. So, in short, response time is less if data is found in cache and response time is more if data is not found in cache.

Caching Size /Experiment	Lookup
1	0.008995542526245117
2	0.008233489990234375
3	0.0067631244659

	423825
4	0.006723995208740234
5	0.005152621269226074
6	0.004201858043670654
7	0.003785469532012939 4
100	0.003589234352111816 4
1000	0.003966214656829834

Design Trade-offs

- Dictionary has been used to perform in-memory caching.
- Caching has only been implemented for improving performance of lookup queries and not search queries.
- Added to the cache like limit on the number of items .
- One replica makes an update to it's database and sends over the details to its counterpart. In this case, the sending replica doesn't check if the changes were made to it's counterpart's database. So, the system doesn't provide a strong guarantee that the databases will be consistent.
- a round-robin algorithm for load balancing mechanism has been used. The load balancer just toggles the request between two servers.


Command Line
Size cache :0
Invalidating cache data for item number :1
192.168.1.123 - - [18/May/2020 08:36:35] "GET /invalidate/1 HTTP/1.1" 200 -
0.046s
192.168.1.208 - - [18/May/2020 08:36:35] "GET /buy/1 HTTP/1.1" 200 -
Client wants to buy book number 1
Sending buy request to the order 1 server.
Size cache :0
Invalidating cache data for item number :1
192.168.1.210 - - [18/May/2020 08:36:35] "GET /invalidate/1 HTTP/1.1" 200 -
0.05s
192.168.1.208 - - [18/May/2020 08:36:35] "GET /buy/1 HTTP/1.1" 200 -
Client wants to buy book number 5
Sending buy request to the order 2 server.
Size cache :0
Invalidating cache data for item number :5
192.168.1.123 - - [18/May/2020 08:36:35] "GET /invalidate/5 HTTP/1.1" 200 -
0.052s
192.168.1.208 - - [18/May/2020 08:36:35] "GET /buy/5 HTTP/1.1" 200 -
Average time search after doing 100 sequential buys is :0.13499710083007813 seconds
192.168.1.208 - - [18/May/2020 08:36:35] "GET /test_buy/100/sizeCached/4 HTTP/1.1" 200 -
Mozilla Firefox
average_time_taken: 0.13499710083007813
numbersTest: 100