Design Document:

This document contains following:

1. **Over View of the Design**
2. **Explanation of how the following is implemented:**
    a. **Replication and Caching**
        i. **Load Balancing**
        ii. **Invalidation**
        iii. **Write Consistency (sync within replica databases))**
    b. **Dockerize your application**
    c. **Fault Tolerance**
3. **Code Structure + Screenshot of application Running**
4. **Experiments (All experiments are perfomed on Elnux machine in distributed environement)**
    a. **Average Response Time per client for each operation**
        i. **With Caching**
        ii. **Without Caching**
        iii. **Comparison of performance wrt above two**
    b. **Average Response time for each request for each micro service**
        i. **With Caching**
        ii. **Without Caching**
        iii. **Comparison of performance wrt above two**
    c. **Invalidation while Caching {ID and topic}**
        i. **Cache Miss**
        ii. **Write Consistency (database write to one replica is in sync with another)**
    d. **Fault Tolerance + Process of Resync and Recovery Screenshots**
    e. **Dockerize Screenshots**

Overview: In this design, there are three servers: Front End, Catalog, Order. The database file maintained is catalog.json. Also, each server (including replicas) maintains its own separate log file. The client interacts with Front End Server and makes requests for lookup, search and buy. The Front End Server on receiving the request from the client interacts with the catalog or order server (as per the request received from the client).

we have used Flask as the backend for implementing the servers (which is also responsible for handling **concurrent requests**) and requests package for maintaining the communication between all the servers. The load balancing is done using Round Robin Method (See below for more info). Also, for the heartbeat mechanism, we have used BackgroundScheduler from flask apscheduler scheduler to run in the background inside our application from Flask

---

1. **Replication and Caching (Along with Load Balancing + Invalidation + Write Consistency (sync within replica databases))**
    a. In this design, there are 2 catalog servers and 2 order servers and 1 Front End Server. The catalog and order servers are differentiated using **server ids indexed with 0 and 1** {provided as input while running the server}.
    b. For **Caching**, we are maintaining an in-memory database {**Key-Value Store**: in the form of a dictionary} which stores all the books data {same as in the catalog.json format}. The internal function calls are being implemented for putting and getting the data from in-memory database. In case, a query comes {search, lookup}, the in-memory database is first searched based on the query received {id, item} and then the corresponding results are immediately returned.
    c. For **load balancing**, we have used **Round Robin Algorithm**. Using this method, client requests are routed to available catalog or order servers on a **cyclical basis**. The frontend directs the query to the catalog and order server.
        i. For search() -> Load Balancing is done using retrieveCatalogID [GET]
        ii. For lookup() -> Load Balancing is done using retrieveCatalogID [GET]
        iii. For buy() -> Load Balancing is done using retrieveOrderID [GET]
    d. For **Invalidating**, with every update call in the catalog server, caches are invalidated calling a REST api at the frontend server.
    e. The design also makes sure that the writes made to one replica is in sync with the other replica (Proof Shown below)

2. **Dockerize your application**
    a. First, the images are built for each server and the client using the docker files. The docker files are present in the src folder. It is implemented using python3.6-Alpine which takes less space than the linux one.
    b. Created a subnet for communication between the docker containers [172.18.0.0/16]
    c. There is a separate config file for docker.
    d. All the steps are mentioned in dockerRun.txt file in src folder
    e. Docker Images are in drive {https://drive.google.com/open?id=1XcYL8OLeRbrLxT9xET8BIO2srqMTANbe}
3. **Fault tolerance**
    a. Used the ping mechanism which is implemented in the Front End Server using the BackgroundScheduler from flask apscheduler which keeps on running in the background. We are using the Round Round Scheduling Algorithm for the load balancing. So, using this, upon detection of failure, the request is routed to the other replica. The front end keeps on checking on all the servers {2 catalog, 2 order} states. For instance, in case server id 0 of catalog server is crashed, then the requests are directed only to the catalog server id 1 and the database of id 0 is sent to the catalog id 1. Meanwhile, front end keeps on pinging the catalog server 0 as well, waiting for it to be back up.

Code Structure: All the python scripts, docker files, input files are placed in the src folder.

1. catalog.json : contains the books database

2. catalog_server.py: contains the code for catalog server – update (POST), query (GET), invalidation of cache is done with the update operation {based on item id or item topic}
3. order_server.py: contains the code for order server – buy (GET)
4. front_end.py: contains the code for front end server – search (GET), lookup(GET), buy(GET), invalidate(GET)
5. createconfig.py: used for creating the config file in the following format:
    a. server_name, ip, port
6. average_response_time_calculator.py : For generating performance metric data + graphs

---

Experimentation (All Experiments are performed on Elnux Machines) with the following config:

Catalog 1 -> Elnux 1 , Catalog 2 –> Elnux 2, FrontEnd –> Elnux 3, Order 1 -> Elnux 7

Order 2 -> Elnux 7, Client – Elnux 3

1. On running the run_2.sh script {bash run_2.sh}, all the dependencies are installed and then prompted for the machine number {elnux 1, 2, 3, 7} for running each server. And then the server starts serving the requests from the client.

```
----------------------------------------------------------------------
Creating the Configuration File
[Enter the first edlab Machine to host the Frontend Server (1,2,3,7): 1          ]
[Enter the second edlab Machine to host the Catalog Server – 1 (1,2,3,7): 1      ]
[Enter the second edlab Machine to host the Order Server (1,2,3,7): 3            ]
[Enter the first edlab Machine to host the Catalog Server-2 (1,2,3,7): 7         ]
[Enter the first edlab Machine to host the Order Server-2 (1,2,3,7): 2           ]
frontend,elnux1,36001
catalog,elnux1,36003
order,elnux3,36011
catalog1,elnux7,36013
order1,elnux2,36017
Front End Server is running on:     elnux1  ------>  36001
Catalog Server – 1 is running on:     elnux1  ------>  36003
Order Server – 1 is running on:     elnux3  ------>  36011
Catalog Server – 2 is running on:     elnux7  ------>  36013
Order Server – 2 is running on:     elnux2  ------>  36017
Running the MicroServices as per the configuration provided
```

Logs for above Configuration: Following logs shows all requests serverd by the servers, caching, invalidation, Heartbeats

a. Front End Logs
    a. If you'll observe, there is a print "Getting Heartbeats": front end periodically checks the catalog and order server
    b. "Checking Cache" – which indicates that the item is found in cache.

```
Getting Heartbeats Response from all the server:

Searching for  DistributedSystems
128.119.243.168 - - [27/Apr/2020 03:39:52] "GET /search?topic=DistributedSystems HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Lookup for  Cooking for the Impatient Graduate Student
128.119.243.168 - - [27/Apr/2020 03:39:57] "GET /lookup?item=4 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Searching for  GraduateSchool
128.119.243.168 - - [27/Apr/2020 03:40:02] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:40:07] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Lookup for  RPCs for Dummies
128.119.243.168 - - [27/Apr/2020 03:40:12] "GET /lookup?item=2 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Buying for  Xen and the Art of Surviving Graduate School
```

```
Searching for  GraduateSchool
128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  GraduateSchool
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Searching for  DistributedSystems
128.119.243.168 - - [27/Apr/2020 03:39:46] "GET /search?topic=DistributedSystems HTTP/1.1" 200 -
Searching for  DistributedSystems
Checking Cache -- Found
```

b. Catalog Logs – For both Replicas (Server – 1 (left) , Server – 2(right))
   a. These logs clearly show that the load balancing is done (See the prints for querying for items)
   b. Write consistency is also shown – If you will observe, "RPC for dummies" (look at the bottom of logs) is first directed to the server -1 (Query Successful print is there) and is then invalidated for the other server (replica)

```
128.119.243.168 - - [27/Apr/2020 03:39:44] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /heartbeat HTTP/1.1" 200 -
Querying for GraduateSchool
Books Data:     [{'stock': 995, 'title': 'How to get a good grade in 677 in 20 minutes a da
, 'cost': 200, 'topic': 'DistributedSystems'}, {'stock': 996, 'title': 'Xen and the Art of
g for the Impatient Graduate Student', 'id': 4, 'cost': 400, 'topic': 'GraduateSchool'}, {'
'stock': 1000, 'title': 'Why Theory Classes are so hard', 'id': 6, 'cost': 600, 'topic': 'G
tributedSystems'}]
Query successful!
128.119.243.168 - - [27/Apr/2020 03:39:45] "GET /query?topic=GraduateSchool HTTP/1.1" 200 -
Querying for How to get a good grade in 677 in 20 minutes a day
Query successful!
128.119.243.168 - - [27/Apr/2020 03:39:46] "GET /query?item=1 HTTP/1.1" 200 -
Querying for Cooking for the Impatient Graduate Student
Query successful!
128.119.243.168 - - [27/Apr/2020 03:39:46] "GET /query?item=4 HTTP/1.1" 200 -
Updating for  Cooking for the Impatient Graduate Student
Update successful!
Invalidating:      Cooking for the Impatient Graduate Student
Invalidating Items: (wrt item id and topic)
128.119.243.164 - - [27/Apr/2020 03:39:46] "POST /update?item=4 HTTP/1.1" 200 -
Querying for RPCs for Dummies
Query successful!
128.119.243.175 - - [27/Apr/2020 03:39:46] "GET /query?item=2 HTTP/1.1" 200 -
Updating for  RPCs for Dummies
Update successful!
Invalidating:      RPCs for Dummies
Invalidating Items: (wrt item id and topic)
128.119.243.175 - - [27/Apr/2020 03:39:46] "POST /update?item=2 HTTP/1.1" 200 -
Updating for  RPCs for Dummies
Update successful!
Invalidating:      RPCs for Dummies
Invalidating Items: (wrt item id and topic)
128.119.243.164 - - [27/Apr/2020 03:39:46] "POST /update?item=2 HTTP/1.1" 200 -
Querying for Cooking for the Impatient Graduate Student
Query successful!
```

```
128.119.243.168 - - [27/Apr/2020 03:40:00] "GET /heartbeat HTTP/1.1" 200 -
Querying for DistributedSystems
Books Data:     [{'id': 1, 'cost': 100, 'title': 'How to get a good grade in 677 in 20 minutes a
 'stock': 995, 'topic': 'DistributedSystems'}, {'id': 3, 'cost': 300, 'title': 'Xen and the Art
 'Cooking for the Impatient Graduate Student', 'stock': 996, 'topic': 'GraduateSchool'}, {'id':
'id': 6, 'cost': 600, 'title': 'Why Theory Classes are so hard', 'stock': 1000, 'topic': 'Gradua
tributedSystems'}]
Query successful!
128.119.243.168 - - [27/Apr/2020 03:40:01] "GET /query?topic=DistributedSystems HTTP/1.1" 200 -
Querying for Xen and the Art of Surviving Graduate School
Query successful!
128.119.243.168 - - [27/Apr/2020 03:40:01] "GET /query?item=3 HTTP/1.1" 200 -
Querying for Cooking for the Impatient Graduate Student
Query successful!
128.119.243.175 - - [27/Apr/2020 03:40:01] "GET /query?item=4 HTTP/1.1" 200 -
Updating for  Cooking for the Impatient Graduate Student
Update successful!
Invalidating:      Cooking for the Impatient Graduate Student
Invalidating Items: (wrt item id and topic)
128.119.243.175 - - [27/Apr/2020 03:40:01] "POST /update?item=4 HTTP/1.1" 200 -
Updating for  RPCs for Dummies
Update successful!
Invalidating:      RPCs for Dummies
Invalidating Items: (wrt item id and topic)
128.119.243.147 - - [27/Apr/2020 03:40:01] "POST /update?item=2 HTTP/1.1" 200 -
Querying for RPCs for Dummies
Query successful!
128.119.243.175 - - [27/Apr/2020 03:40:01] "GET /query?item=2 HTTP/1.1" 200 -
Updating for  RPCs for Dummies
Update successful!
Invalidating:      RPCs for Dummies
Invalidating Items: (wrt item id and topic)
128.119.243.175 - - [27/Apr/2020 03:40:01] "POST /update?item=2 HTTP/1.1" 200 -
Updating for  Cooking for the Impatient Graduate Student
Update successful!
Invalidating:      Cooking for the Impatient Graduate Student
Invalidating Items: (wrt item id and topic)
```

c. Order Logs

```
Querying for Cooking for the Impatient Graduate Student
Bought  Cooking for the Impatient Graduate Student
128.119.243.168 - - [27/Apr/2020 03:46:50] "GET /buy?item=4 HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:46:54] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:46:55] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:46:59] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:00] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:04] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:05] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:09] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:10] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:14] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:15] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:19] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:20] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:24] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:25] "GET /heartbeat HTTP/1.1" 200 -
Querying for Cooking for the Impatient Graduate Student
Bought  Cooking for the Impatient Graduate Student
```

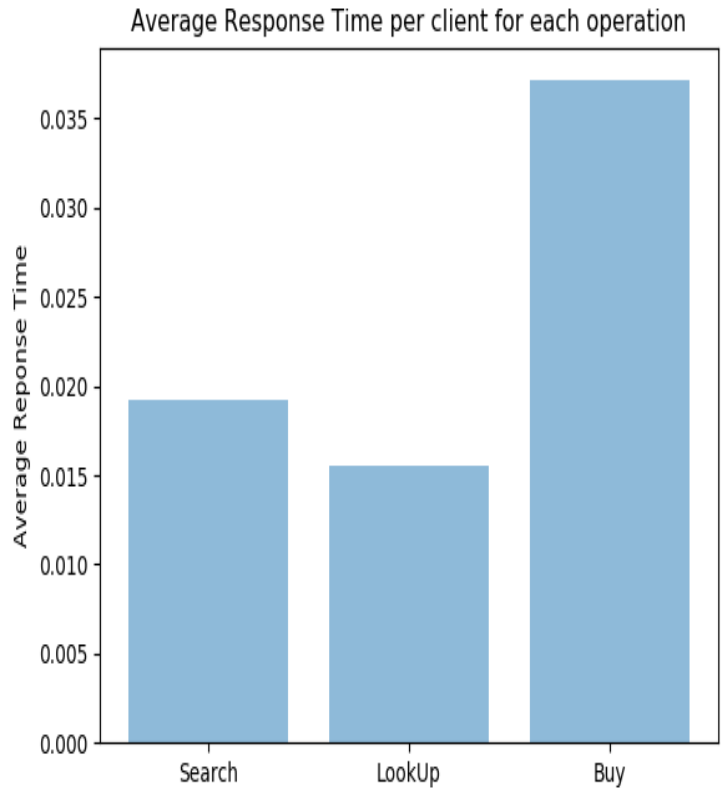
```
Bought  Xen and the Art of Surviving Graduate School
128.119.243.168 - - [27/Apr/2020 03:39:47] "GET /buy?item=3 HTTP/1.1" 200 -
Querying for Xen and the Art of Surviving Graduate School
Bought  Xen and the Art of Surviving Graduate School
128.119.243.168 - - [27/Apr/2020 03:39:47] "GET /buy?item=3 HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:39:49] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:39:50] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:39:54] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:39:55] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:39:59] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:00] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:04] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:05] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:09] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:10] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:14] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:15] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:19] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:20] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:24] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:25] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:29] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:30] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:34] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:40:35] "GET /heartbeat HTTP/1.1" 200 -
Querying for How to get a good grade in 677 in 20 minutes a day
Bought  How to get a good grade in 677 in 20 minutes a day
```

d. Client Logs

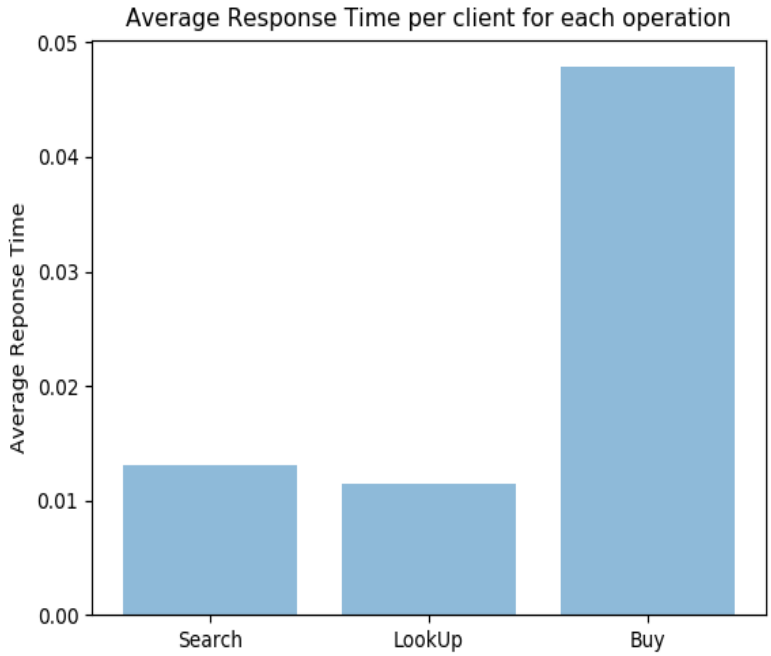
```
buy
Trying to buy Xen and the Art of Surviving Graduate School
200
Successfully bought Xen and the Art of Surviving Graduate School
search
Starting search for DistributedSystems
200
{
    "books": [
        {
            "cost": 100,
            "id": 1,
            "stock": 994,
            "title": "How to get a good grade in 677 in 20 minutes a day",
            "topic": "DistributedSystems"
        },
        {
            "cost": 200,
            "id": 2,
            "stock": 992,
            "title": "RPCs for Dummies",
            "topic": "DistributedSystems"
        },
        {
            "cost": 700,
            "id": 7,
            "stock": 1000,
            "title": "Spring in the Pioneer Valley",
            "topic": "DistributedSystems"
        }
    ]
}
lookup
Starting lookup for Cooking for the Impatient Graduate Student
```

---

2. **Average Response Time per client for each operation (Performance Comparison with Caching and Without Caching)**:
    a. **Without Caching** {For 1000 requests per client}



b. **With Caching** {For 1500 requests per client}



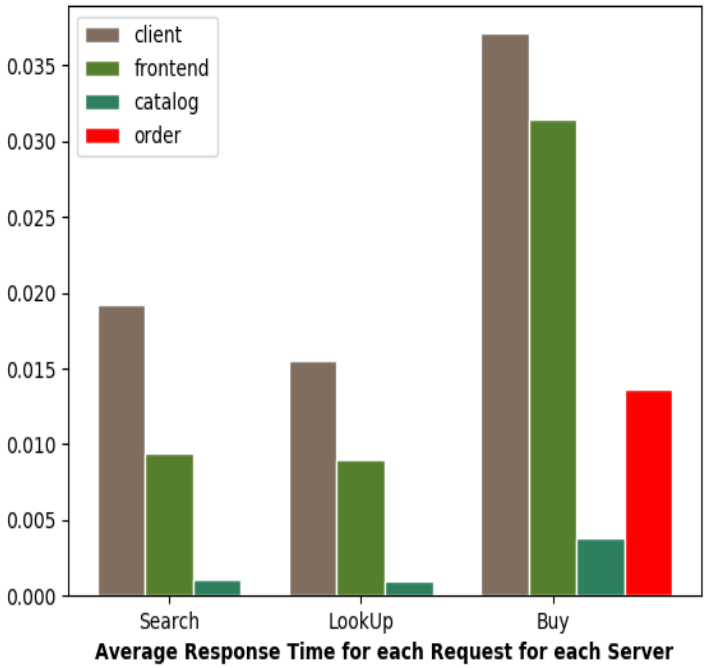
Observation:

1. The above 2 graphs clearly show that the search() and look up () time has been reduced after introducing caching.
2. Percentage reduction in search and lookup query average response time with caching is around 45-50%.
3. Buy Query takes more time than the search and lookup query in both the cases
4. Search and Lookup Query takes almost the same time which is desirable also.

	Average Response Time Per Client (in Sec) – without caching {for 3000 req}	Average Response Time Per Client (in Sec) – with caching {for 5000 requests}
Search()	0.020 s	0.013s
Lookup()	0.017 s	0.01s
Buy()	0.038 s	0.048s

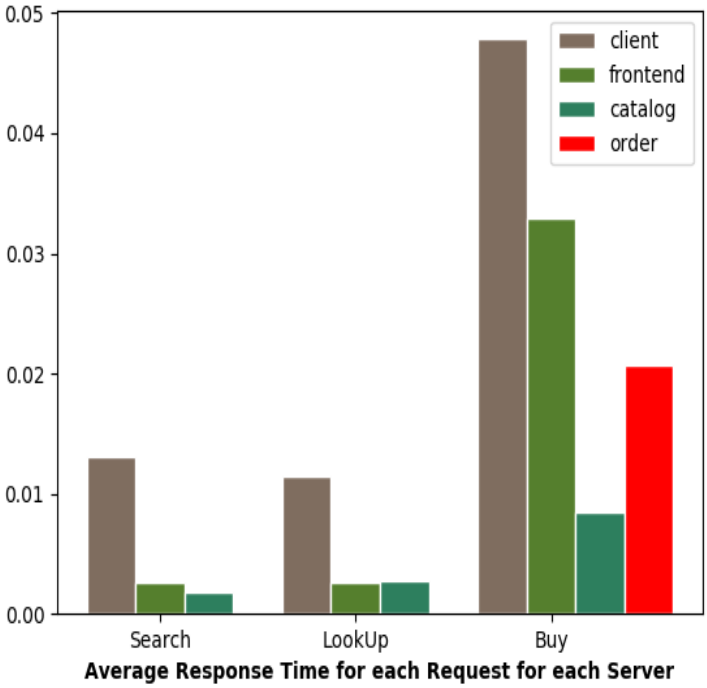
3. Average Response time for each request for each micro service (Performance Comparison with Caching and Without Caching)

a. Without Caching



Average Response Time for each Request for each Server

b. With Caching



Observation:

1. Search and lookup response times have been reduced after introducing caching (to about 50%). However, the buy response time in case of caching and without caching is almost same.
    a. In-memory database(key-value) holds used data/request which may be requested next and it is faster access memory than the persistent database (RAM). This reduces the need for slower memory retrievals from main memory, which may otherwise keep the CPU waiting.
2. Search and lookup time is 0 in case of order Server as it is responsible for just buy request – whether an item is bought successfully or is it out of stock
3. Time taken is more in case of catalog server than the other two because of there is no network involved at the catalog server. Time increase from catalog to frontend to client because of network latency.

a. **Without Caching** {for 1000 requests}

	Client(s)	Frontend(s)	Catalog(s)	Order(s)
search	0.020	0.010	0.02	0
lookup	0.017	0.010	0.02	0
buy	0.038	0.034	0.06	0.014

b. **With Caching** {for 1500 requests} {Caching Logs and Proof is shown in above screenshots)}

	Client(s)	Frontend(s)	Catalog(s)	Order(s)
search	0.013	0.003	0.002	0
lookup	0.01	0.003	0.003	0
buy	0.048	0.034	0.06	0.018

Therefore, percentage reduction in average response time in search and lookup is approximately {for 1000 requests}, 45-50% for each.

---

4. Invalidation while Caching {ID and topic} & Write Consistency {Invalidation Proof is shown in above screenshots}

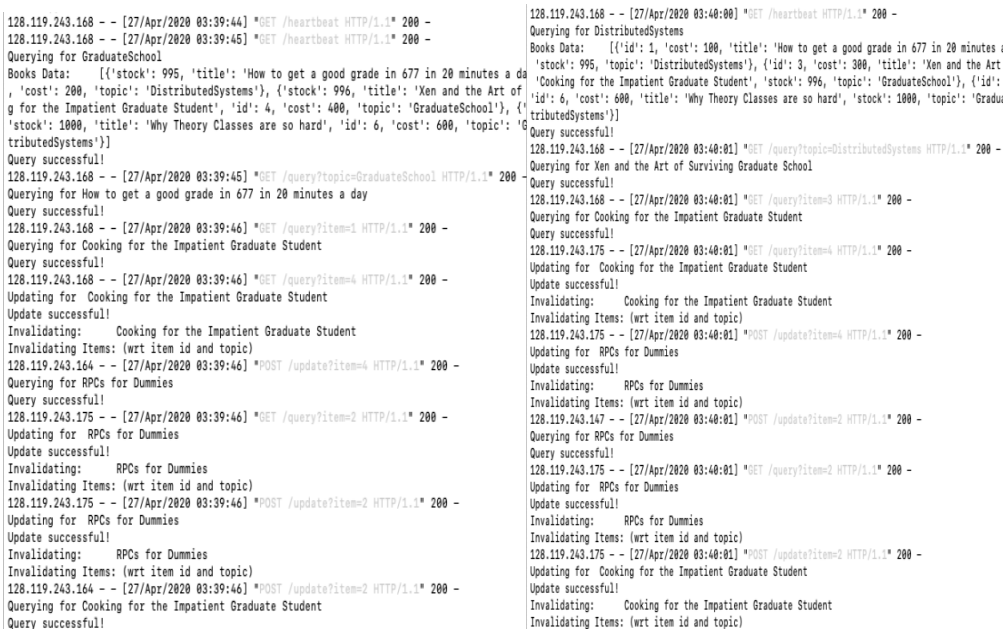
With every update api in catalog server, invalidation of the item is done. In order to observer the perform performance observations, we just call the **buy**() request at the client side, which in turn calls for the update api in the catalog server. We performed experiments for both the cases when the invalidation is done and when it is not done and calculated the average response times. The results are shown below.

	Client(s)	Frontend(s)	Catalog(s)	Order(s)
Invalidation is done	0.013	0.006	0.004	0.005
Invalidation is not done	0.012	0.0057	0.0028	0.035

a. As the results are shown above, the **invalidation incurs some amount of overhead in case of servers and the client**.
b. The overhead incurred in client, frontend, catalog, order is around 7%, 6%, 30%, 6% respectively. The most amount of overhead is observed in the **catalog** server is around **4 times more than the others**.
c. Also, **we observed that the cache miss** (latency of request after invalidation is introduced) **is 0.013 sec for buy() request, around 0.003 sec for both lookup() and search() calls**.

Avoiding replication of write to the Replica Server (Maintaining write consistency):

** If you will observe in the below screenshots (catalog server – 1(left) , catalog server-2 (right)), "RPC for dummies" (look at the bottom of logs) is first directed to the server -1 (Query Successful print is there) and is then invalidated for the other server (replica)



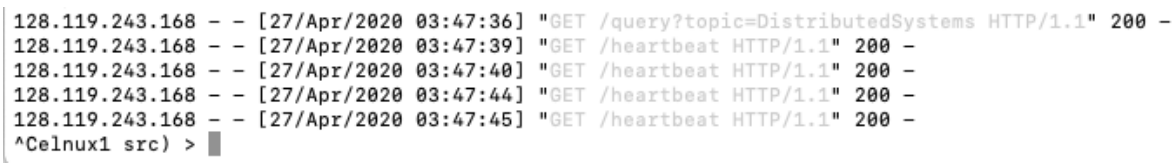
For avoiding write replication in case of update call in catalog server, POST calls are made from one replica to the another and during this a flag is maintained which ensures that if the flag is set to 1, then only the POST request for write is sent to the replica, else it does not. This way, replication among different replicas is maintained and the write consistency is ensured.

---

5. Fault Tolerance + Process of Resync and Recovery

For ensuring Fault Tolerance, we are making use of heartbeat mechanism. The frontend server periodically sends heartbeat to the order and the catalog server. This way, frontend, keeps track of each server {which is running or crashed} using the GET requests. The round robin approach has been followed.

1. Catalog Server – 1 is Stopped



2. Front End Logs (“Catalog Server – 1 is not Running”)

```
128.119.243.168 - - [27/Apr/2020 03:47:41] "GET /lookup?item=2 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Searching for  DistributedSystems
Checking Cache -- Found

128.119.243.168 - - [27/Apr/2020 03:47:46] "GET /search?topic=DistributedSystems HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Catalog Server:1 is Not Running
Getting Heartbeats Response from all the server:

Catalog Server:1 is Not Running
Searching for  GraduateSchool
128.119.243.168 - - [27/Apr/2020 03:47:51] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Catalog Server:1 is Not Running
Getting Heartbeats Response from all the server:

Catalog Server:1 is Not Running
Buying for  How to get a good grade in 677 in 20 minutes a day
128.119.243.175 - - [27/Apr/2020 03:47:56] "GET /retrieveCatalogID HTTP/1.1" 200 -
Invalidating the item:  How to get a good grade in 677 in 20 minutes a day
128.119.243.164 - - [27/Apr/2020 03:47:56] "GET /checking?item=1 HTTP/1.1" 200 -
Invalidating the item:  DistributedSystems
128.119.243.164 - - [27/Apr/2020 03:47:56] "GET /checking?topic=DistributedSystems HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:47:56] "GET /buy?item=1 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:
```

3. Catalog Server – 2 logs (serving the requests)

```
Query successful!
128.119.243.168 - - [27/Apr/2020 03:48:42] "GET /query?topic=DistributedSystems HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:48:45] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 03:48:45] "GET /heartbeat HTTP/1.1" 200 -
Querying for RPCs for Dummies
Query successful!
128.119.243.175 - - [27/Apr/2020 03:48:47] "GET /query?item=2 HTTP/1.1" 200 -
Updating for  RPCs for Dummies
```

Process of Resync and Recovery

For this, we maintained an input argument to the serve which states that the server is starting for the very first time or is it in recovery mode. Using this, we initialized out server as the fresh server or using the replica, incase it's crashed. While starting the server, the REST endpoint checks if the catalog server has been crashed in the past. If yes, then it is resynced with its replica server.

```
Resyncing the Server

Transferring to the other catalog server

 * Serving Flask app "catalog_server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://elnux1:36003/ (Press CTRL+C to quit)
 * Restarting with stat
Resyncing the Server

 * Debugger is active!
 * Debugger PIN: 201-833-257
128.119.243.168 - - [27/Apr/2020 04:21:40] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:41] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:45] "GET /heartbeat HTTP/1.1" 200 -
```

```
(elnux2 src) > python3 catalog_server.py 0 1
 * Serving Flask app "catalog_server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://elnux2:36013/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 281-686-654
128.119.243.168 - - [27/Apr/2020 04:21:30] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:31] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:35] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:36] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:40] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:41] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:45] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:46] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:50] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:51] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.147 - - [27/Apr/2020 04:21:51] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.147 - - [27/Apr/2020 04:21:51] "GET /reassign HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:55] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:21:56] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:22:00] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:22:01] "GET /heartbeat HTTP/1.1" 200 -
128.119.243.168 - - [27/Apr/2020 04:22:05] "GET /heartbeat HTTP/1.1" 200 -
```

```
Catalog Server:1 is Not Running
128.119.243.147 - - [27/Apr/2020 04:21:36] "GET /notRunning?id=0 HTTP/1.1" 200 -
128.119.243.147 - - [27/Apr/2020 04:21:36] "GET /notRunning?id=0 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:

Getting Heartbeats Response from all the server:
```

---

4. **Dockerize the Application – All Steps are in dockerRun.txt**
    a. Build the images from all the docker files {dockerfile_order, dockerfile_frontend, dockerfile_catalog, dockerfile_client}
    b. Created a subnet
    c. Run the servers using -> For example: docker run --net sub --ip 172.18.1.1 -p 5213 frontend
    d. There is a separate config file for docker {configDocker.txt}

```
Step 4/9 : RUN pip install --upgrade pip
 ---> Running in be721d792774
Requirement already up-to-date: pip in /usr/local/lib/python3.6/site-packages (2
0.0.2)
Removing intermediate container be721d792774
 ---> 7773c6a4247a
Step 5/9 : RUN pip install flask
 ---> Running in 738c894c7d48
Collecting flask
  Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
Collecting Jinja2>=2.10.1
  Downloading Jinja2-2.11.2-py2.py3-none-any.whl (125 kB)
Collecting click>=5.1
  Downloading click-7.1.1-py2.py3-none-any.whl (82 kB)
Collecting Werkzeug>=0.15
  Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
Collecting itsdangerous>=0.24
  Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=0.23
  Downloading MarkupSafe-1.1.1.tar.gz (19 kB)
Building wheels for collected packages: MarkupSafe
  Building wheel for MarkupSafe (setup.py): started
```

src > configDocker.txt

```
1  frontend,172.18.1.1,5213
2  catalog,172.18.1.2,5211
3  order,172.18.2.1,5212
4  catalog1,172.18.2.2,5214
5  order1,172.18.3.1,5215
```

dockerfile_frontend

```
FROM python:3.6-alpine
COPY . /src
WORKDIR /src
RUN pip install --upgrade pip
RUN pip install flask
RUN pip install requests
RUN pip install apscheduler


RUN mv configDocker.txt config.txt
ENTRYPOINT ["python", "-u", "front_end.py"]
```

```
172.18.1.2 - - [27/Apr/2020 08:46:03] "GET /checking?item=3 HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:03] "GET /checking?topic=GraduateSchool HTTP/1.1" 200 -
172.18.2.2 - - [27/Apr/2020 08:46:03] "GET /checking?item=3 HTTP/1.1" 200 -
172.18.2.2 - - [27/Apr/2020 08:46:03] "GET /checking?topic=GraduateSchool HTTP/1.1" 200 -
172.18.1.4 - - [27/Apr/2020 08:46:03] "GET /buy?item=3 HTTP/1.1" 200 -
172.18.1.4 - - [27/Apr/2020 08:46:03] "GET /search?topic=DistributedSystems HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

172.18.1.4 - - [27/Apr/2020 08:46:08] "GET /lookup?item=1 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

172.18.2.1 - - [27/Apr/2020 08:46:13] "GET /retrieveCatalogID HTTP/1.1" 200 -
172.18.2.2 - - [27/Apr/2020 08:46:13] "GET /checking?item=2 HTTP/1.1" 200 -
172.18.2.2 - - [27/Apr/2020 08:46:13] "GET /checking?topic=DistributedSystems HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:13] "GET /checking?item=2 HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:13] "GET /checking?topic=DistributedSystems HTTP/1.1" 200 -
172.18.1.4 - - [27/Apr/2020 08:46:13] "GET /buy?item=2 HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:

172.18.1.4 - - [27/Apr/2020 08:46:18] "GET /search?topic=GraduateSchool HTTP/1.1" 200 -
Getting Heartbeats Response from all the server:
```

```
Ritus-MacBook-Air:src ritu$ docker run --net sub --ip 172.18.1.4 client
1        2         0
http://172.18.1.2:5211/
http://172.18.2.1:5212/
http://172.18.1.1:5213/
search
Starting search for DistributedSystems
200
{
    "books": [
        {
            "cost": 100,
            "id": 1,
            "stock": 993,
            "title": "How to get a good grade in 677 in 20 minutes a day",
            "topic": "DistributedSystems"
        },
        {
            "cost": 200,
            "id": 2,
            "stock": 994,
            "title": "RPCs for Dummies",
            "topic": "DistributedSystems"
```

```
172.18.1.1 - - [27/Apr/2020 08:46:00] "GET /heartbeat HTTP/1.1" 200 -
172.18.1.1 - - [27/Apr/2020 08:46:01] "GET /heartbeat HTTP/1.1" 200 -
172.18.1.1 - - [27/Apr/2020 08:46:02] "GET /query?topic=DistributedSystems HTTP/1.1" 200 -
172.18.1.1 - - [27/Apr/2020 08:46:02] "GET /query?item=4 HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:02] "POST /update?item=3 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "GET /query?item=4 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "POST /update?item=4 HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:02] "POST /update?item=1 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "GET /query?item=2 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "POST /update?item=2 HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:02] "POST /update?item=4 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "GET /query?item=4 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "POST /update?item=4 HTTP/1.1" 200 -
172.18.1.2 - - [27/Apr/2020 08:46:02] "POST /update?item=3 HTTP/1.1" 200 -
172.18.3.1 - - [27/Apr/2020 08:46:02] "GET /query?item=4 HTTP/1.1" 200 -
```