An-Najah National University

Distributed Operation Systems
Dr. Samer Arandi
Bazar
Fall-Semester 2022

Student Name	Student ID
Essa Waleed Abbadi	11819883
Ahmad Hamad Qorom	11821492

Bazar

✓ Program Design

This design is constructing from five micro-services the front-end, two catalog micro-services and two order micro-services.

The front-end will accept user requests and perform load balancing using **round-robin** algorithm and caching using cache replacement policy **LRU**.

While the catalog micro-services maintains the catalog database with new three books:

1-How to finish Project 3 on time
2-Why theory classes are so hard.
3-Spring in the Pioneer Valley

The order micro-services maintains a list of all orders received for the books (it contains the id and timestamp).

The **search(topic)** and info **(item_number)** are read operations so there response is stored in the in-memory cache in front-end micro-service if not there in advanced.

while **purchase(item_number)** is changing on the catalog and order Databases so it perform a number of requests for achieving consistency. We use consistency with **Primary-backup remote-write protocol** as Catalog1 is the primary-backup for the two catalog micro-services and Order1 is the primary-backup for the two order micro-services.

✓ How it works

it works in five different virtual machines or PCs as each service is separated from other services and has its own database, they interact with each other by sending HTTP requests and received response in JSON formats.

The client uses the browser/postman to send HTTP requests to the front end micro-service and it will do the request using a load balancing algorithm **round-robin** by interacted with other micro-services and replay with response to the client or replay from the in-memory cache if the response is saved.

Design trade-offs

We use **round-robin** for load balancing algorithm instead of **least loaded** because it is easier to implement in small systems and the queries have almost the same load except of purchase query which is not common.

Also we use **in-memory cache** instead of **separate component** because it is faster and the cache in general in small. Because we implement a limit on the number of items in the cache with replacement policy **LRU**.

In addition, in consistency we use **Primary-backup remote-write protocol** instead of **Primary-backup local-write protocol** because the read queries are more frequent than the write queries.

Possible Improvements and extensions

We can improve the design by make the micro-services search for each other IP's dynamically without the need to put static IP's. Also we can do the same for port numbers.

We can add another catalog and order micro-services that do the same job and interact with the front-end micro-service.

When it will not work

It will not work in these cases:

1- There is no connection between the micro-services.
2- There are too many requests to the database at the same time.
3- The IP for the other micro-services is wrong.
4- The path to the database is wrong.

Steps to run the program

First step: put the IP and port number for the catalog and orders micro-services in the strings below in the "FrontMain.java".

```
public static String catalogIP_Port1 = "localhost:8077"; // ip and port for  catalog1 microservice
public static String orderIP_Port1 = "localhost:8088"; // ip and port for  order1 microservice

public static String catalogIP_Port2 = "localhost:8078"; // ip and port for  catalog2 microservice
public static String orderIP_Port2 = "localhost:8089"; // ip and port for  order2 microservice
```

Also, put the corresponding IP and port numbers in these files:

"OrderMain.java"

```
public static String catalogIP_Port = "localhost:8077"; // ip and port for  catalog1 microservice
public static String orderIP_Port2 = "localhost:8089"; // ip and port for  order2 microservice
```

"OrderMain2.java"

```
public static String catalogIP_Port = "localhost:8078"; // ip and port for  catalog2 microservice
public static String orderIP_Port1 = "localhost:8088"; // ip and port for  order1 microservice
```

"CatalogMain.java"

```
public static String catalogIP_Port2 = "localhost:8078"; // ip and port for  catalog2 microservice
```

“CatalogMain2.java”

```
public static String catalogIP_Port1 = "localhost:8077"; // ip and port for  catalog1 microservice
```

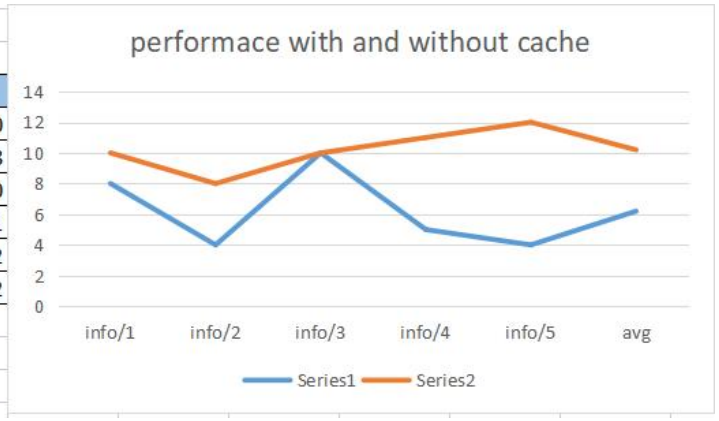
Then, you might create runnable jar files and run them in different machines, or use java IDE to run the micro-services in different machines.

Finally, use browser or postman to send the HTTP requests to the front-end micro-service using the IP and port number for it in the URL.

Conclusion and final Results

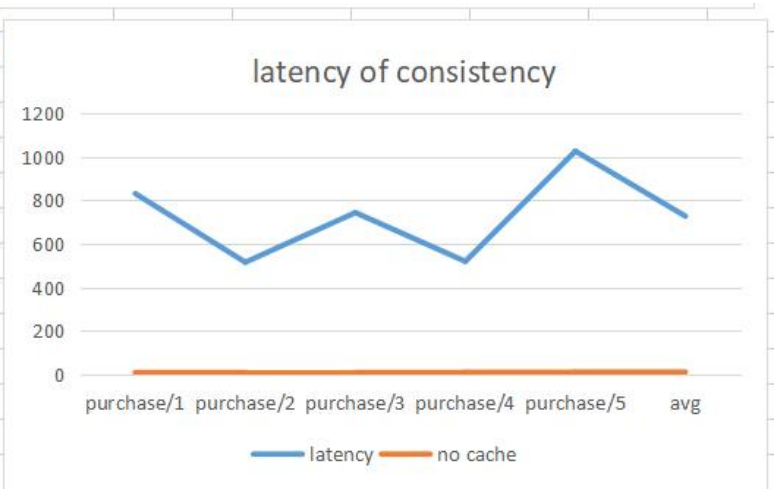
From the experiments we did the performance is improved a lot when using cache as shown below:

	cache	no cache
info/1	8	10
info/2	4	8
info/3	10	10
info/4	5	11
info/5	4	12
avg	6.2	10.2



But the latency of consistency still remain an performance killer (it takes a lot of time) as shown below:

	latency	no cache
purchase/1	831	10
purchase/2	516	8
purchase/3	744	10
purchase/4	520	11
purchase/5	1026	12
avg	727.4	10.2



Unfortunately the consistency is a must in distributed systems but we may use more relaxed consistency in future projects.