Bazar System -> Part_2 :

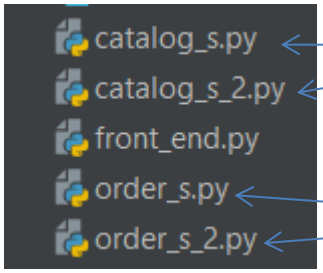
Turning the Bazar into an Amazon: Replication, Caching and Consistency:

Note: in GitHub I provided all the screen shot for all work

First, Bazar.com has three NEW books in its catalog:

```
db.session.add(
    book_s(book_title='How to finish Project 3 on time', book_typee='Graduate School', cost=15,
           number_of_book=11)),
db.session.add(
    book_s(book_title='Why theory classes are so hard', book_typee='Graduate School', cost=35,
           number_of_book=17)),
db.session.add(
    book_s(book_title='Spring in the Pioneer Valley', book_typee='Graduate School', cost=44,
           number_of_book=18)),
db.session.commit()
```

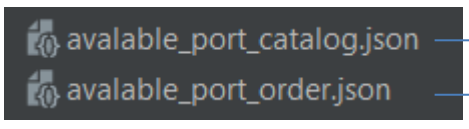
Replication :



catalog server are replicated - their code and their database files

order server are replicated - their code and their database files

load balancing algorithm:



{["available_port_catalog": ["6000", "6002"}

{["available_port_order": ["5000", "5002"}

Note:

Port 6000 for catalog_s.py

Port 6002 for catalog_s_2.py

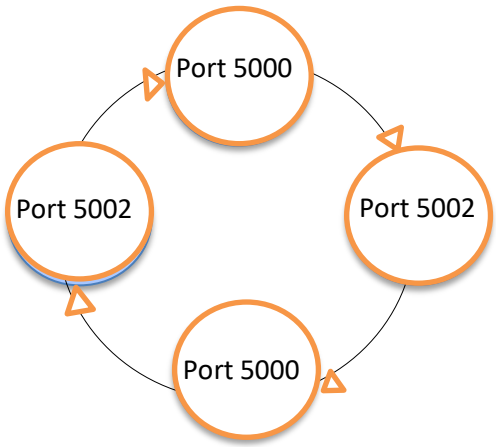
Port 5000 for order_s.py

Port 5002 for order_s_2.py

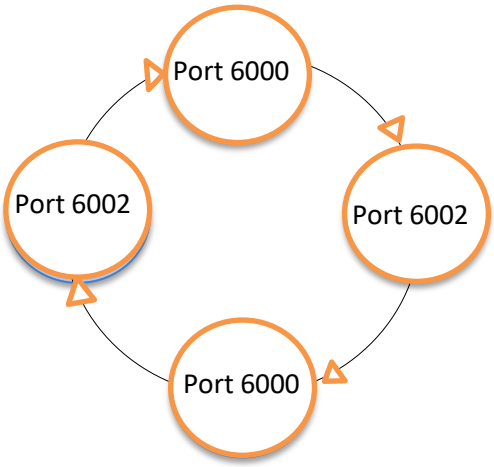
Load balance code:

```
available_servers = open('avalable_port_catalog.json')
data = json.load(available_servers)
port_list1 = list(data["available_port_catalog"])


available_servers = open('avalable_port_order.json')
data = json.load(available_servers)
port_list2 = list(data["available_port_order"])


def next_available_server_order():
    global port_list2
    next_port = port_list2.pop(0)
    port_list2.append(next_port)
    return str(next_port)

def next_available_server_catalog():
    global port_list1
    next_port = port_list1.pop(0)
    port_list1.append(next_port)
    return str(next_port)
```

What load balance (Round Robin) do in chart:



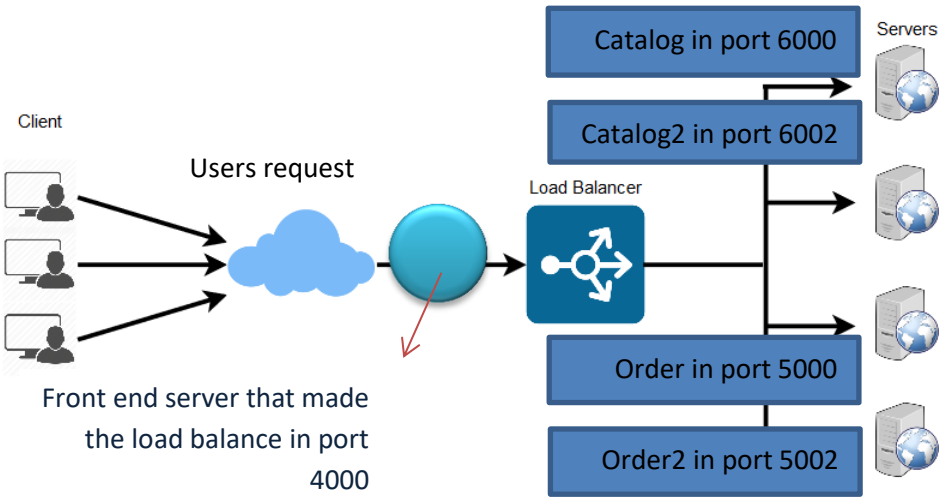
Load balance for order server



Load balance for catalog server

The load balancer:

1. a client sends a request to the load balancer;
2. the load balancer redirects the client to one of the available servers in the pool (catalog server/catalog 3 server / order server/ order 2 server)
3. the server returns the result of the factorial to the client;
4. the server sends a request to the load balancer, advertising that it finished the previous job and is available to serve another client.



When a request arrives to the load balancer, it will get the first available server from that list and redirect the client to said server. This is done by the `next_available_server()` method, which returns the port of the available server and add it the last of list
Then if the same request or different request is send it will go the second available port in list and add it in the last list.

I have implemented it in such way since I believe that this performs better than the classic Round Robin implementation.

The list of available ports is not hardcoded, it is contained in the `avalabile_port_catalog.json` file

The load balancer continues passing requests to servers based on this order. This ensures that the server load is distributed evenly to handle high traffic.

Caching :

Because changes to a book could render an entire topic invalid, it can be more difficult to cache search results. In Bazar, updates don't affect the fields that are returned in search results, so invalidating is not required. Using the search string as the key and the JSON return of the search operation as well as a list of all topics contained in that response, I came up with a solution to this issue. All cached items are examined for their topic set before any subject sets that contain the requested topic are removed from the cache when the catalog server makes an invalidate request for a topic on its end-point in the front-end server. It was added a further endpoint that .

Cashing code in front end server:

```
def query_by_id(book_id):
    return book_s.query.filter(book_s.id == book_id).all()

def query_by_topic(book_typee):
    return book_s.query.filter(book_s.book_typee ==book_typee).all()

queries = {
     'search': {
        'query_by_topic': query_by_topic,
        'schema': BookSchema_search
    },
    'info': {
        'query_by_id': query_by_id,
        'schema': BookSchema_info
    }
}
```

we get data back
out of our database from Flask-SQLAlchemy provides a **query** attribute on your **Model** class. When you access it you will get back a new query object over all records

What happen if the user send the request :

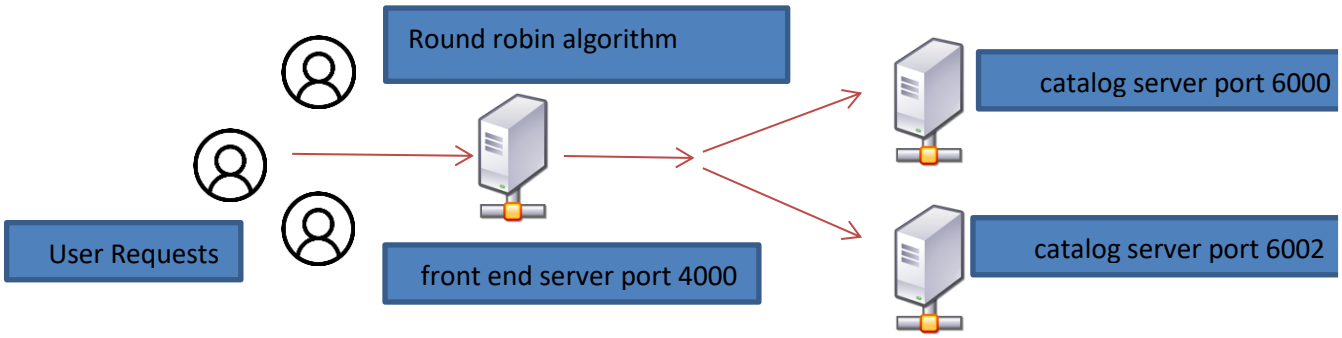
http://127.0.0.1:4000/query/info/1

because the request is about information of book number one

the server who receive the request is catalog server and because we made replicated there is two servers in port 6000 and port 6002

by software code in front end the load balance (round robin send the request

to one of services according to the last one in the list and the server return with the response



Bazar V1 vs Bazar V2 in read operation :

I will be comparing the overall system performance for both Bazar V1 and Bazar V2 running with 2 replicas of order and catalog servers

Bazar V2 is running with 2 replicas for catalog and order servers.

In this test, 100 requests were sent to each front-end server.

Bazar_virsion_one	Bazar_virsion_two
5.32 ms	2.43 ms

Percentage: We can see that the cache had a **45.56%** improvement.

Bazar V1 vs Bazar V2 in write(update) operation

The time that update operation need=

time when check in cash+Time due to replication+ the needed time to update.

That means this case is go worse in the system.

Average response times for 100 book:

Bazar_virsion_one	Bazar_virsion_two
20.11 ms	26.43 ms

Percentage: We can see that performance decrease about **13.14%** .