Distributed operating systems

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

Student: *Ala’a Haj Ali*, آلاء حج علي

Overall Design

- **Frontend service**:
    - In this server all requests from clients are handled and directed to the desired service.
    - This service contains the cache implementation where the cache is a list that maintains dictionary items where each dictionary contains a book details. There are two cache get methods one for the lookup operation to maintain a book info and the other is for the search operation to maintain a list of the books with the searched topic. One set method is built for the cache to maintain items when searching for them the first time from one of the servers. One invalidate method is built in order to remove an item from the cache when this item is modified by one of the write operations.
    - This service also contains the load balancing implementaion where round robin algorithm is adopted, each time a read operation is requested, the request goes to the server that has the turn to perform the request. This switching happens by changing a counter's value each time a request is performed where each value corresponds to one of the servers, in my case this counter is a flag since there is only two servers for each service.
    - Single leader replication is adopted in implementation meaning that read operations can be performed by any server while write operations can only be performed by one server which is called the leader, the leader then sends these write operations or modifications to the rest of the servers.
    - Syncronous replication is also adopted in the implementation meaning that the leader sends the modifications to all the replicas first and then returns the result to the frontend service.
    - In order to maintain availability of the services, if the server that has the turn is down when performing a read operation the request is then sent to the other replica, if all servers are down then the client would receive a failure message.
    - In order to maintain the consistency of the services, if one of the servers that maintain the database is down then any write operation will not be performed until - in my case - both servers are alive and the user will receive a failure message. This can also be maintained using other methods like having a backup server that maintains only the write requests in a database while any server is down and when a server becomes live it will get those modifications from the backup server and when all modifications are performed they will be deleted from this backup server. Either ways there is no guarantee for the consistency since any server can go down even the backup server, but the later method would be more effecient especially if there are more than one replica for each service.
- **Catalog-leader:**
    - This server is the responsible for the write operations and sending modifications to the other replica, if the replica is dead no modifications would be made. It also performs the read operations when it is its turn.
- **Catalog-replica:**

    - This server performs read operations when it is its turn and performs write operations that it gets from the leader.
- **Main-log:**
    - This server maintains the buying service when it is its turn, it reads the book from the server thar has the turn and updates the leader, if one of them is down the buying operation will not be performed and a failure message would be returned to the user.
- **Log-replica:**
    - Same as the main-log server.

How it works

- First step install the requirements from the requirements file to import all needed packages/libraries using the following command:
  pip for windows or *pip3* for ubuntu.

  ```
  >> pip install -r requirments.txt
  ```
- Place each service on a machine and maintain its ip address. In my case, each server was on an ubuntu VM -four ubuntu VMs- except for the frontend server stayed on the host machine -windows 10-. Either ways, place each collected ip address of the servers in the frontend.py file each corresponding to its name.
- Run all servers by running the python script for each using the following command:
  python for windows or *python3* for ubuntu.

  ```
  >> python ‘filename’.py
  ```
- Use postman or other tool as the client to send request to the frontend service using its ip address with the correct request url/name -all are written in the frontend.py- and notice the output for each request.

Performance Analysis

Average Response Time Analysis

Operation/time	Without cache (ms)	With cache (ms)	Improvement percentage:
Search/topic	93.11	41.5	55.43%
lookup/book	98.94	57.39	42%

- The above table shows how the response time of the read requests improves when using the cache since the information will be maintained directly from the same place without redirecting or checking a database.
- When performing a write operation on an item that exists in the cache, the response time for this operation increases since in this case the cache would maintain an invalidation for this item.
- The response time also increases when reading an item that was invalidated or when reading multiple items that are not in the cache - cache miss- since in this case the read request would be directed to one of the catalog servers and then checking its databse for the item/s.