PROJECT REPORT – Designing Two Architecture for My Book stores

This project aims a designing a two tire book store. In which we have a frond end server which simply by-pass the request coming from the client. We have three front end server which can accept communication from several clients. Each of the front end servers will contact with the back end server to process the request from the client. Some of the notable features in this design are ***caching of the Search request*** and ***synchronization among the front end servers***. I have used RPC sockets as a communication protocol.

How Caching for the Search request implemented on the front-end Server?

I have maintained a buffer in each of the front end servers. Each of the server will by-pass the initial search request to the back end server to get the response. Each response is cached in until available information is obtained. So in our case we have two categories of books can be searched. So initial two different request is by-passed and the response is cached in the front end server. This can be verified by checking the number of search request count. Each front end server will contact the back end server at most 2 times. So, in our case we have 3 front end servers, so total search request count at maximum to the back end server will be 6 (two from each of the front end servers).

How synchronization among the front end server is done?

I have used Berkeley algorithm to maintain synchronization among the front end server. In which one of the front end server will act as a master and rest of the two will be slave. So, in the master I have implemented a thread which will periodically adjust the clock to itself and its slaves. The thread will synchronize its slave for every 60 seconds. For the master to synchronize with the slaves: both the slaves have to be started before the master. Then the master is started with the IP address of the two slaves to be synchronized.

The thread function which does the synchronization:

void *synchronize_time(void *arg) – master.c

It calculates the offset by averaging the seconds and milliseconds of the servers to be synchronized and alerts them with the offset to be adjusted to itself and the slaves.

How order request are managed?

Each of the order request are timestamped in the front end servers. When it arrives in the back end server, it is checked whether any pending request are there for comparing the time stamp if it is not there, then the received request is executed freely.

How 100th request from a user is provided discount?

We have a counter which is maintained for the each of the order request. So, the order count is modulated with 100. If the remainder is Zero then the discount message is displayed to the user. **For the purpose of testing I have set the value of 5**. So, it will be easier for you to do the testing. You can also change if you need to configure for the every 100th request to be discounted.

<u>**Steps for changing the discount count for 100 orders:**</u>

1. Go to the SERVER directory. There you will have a header file called **<server.h>**.
2. Inside the header file I have defined a macro called DISC_REQ. change the value to 100 if you want to set the discount for every 100th request.

```
#define DISC_REQ    5
```

Steps to run the project?

```
First Server should be started
------------------------------
Command steps:
-------------
elk02:~/working/OS/SERVER> make
cc server.c -o server -lpthread
elk02:~/working/OS/SERVER> sh run.sh
Server Started....


Next, Slave1 should be started
------------------------------
Command steps:
-------------

elk03:~/working/OS/FRONT_END/SLAVE1> make
cc slave1.c -o slave1 -lpthread
elk03:~/working/OS/FRONT_END/SLAVE1> sh run.sh
Enter the Server IP:
XXX.XXX.XXX.XXX


Next, Slave2 should be started
------------------------------
Command steps:
--------------
elk03:~/working/OS/FRONT_END/SLAVE2> make
cc slave1.c -o slave2 -lpthread
elk03:~/working/OS/FRONT_END/SLAVE2> sh run.sh
Enter the Server IP:
XXX.XXX.XXX.XXX
```

Next, Master should be started

Command steps:

```
elk05:~/working/OS/FRONT_END/MASTER> make
cc master.c -o master -lpthread
elk05:~/working/OS/FRONT_END/MASTER> sh run.sh
Enter Slave1 IP:
XXX.XXX.XXX.XXX
Enter Slave2 IP:
XXX.XXX.XXX.XXX
Enter the Server IP:
XXX.XXX.XXX.XXX
Master started....
```

Finally, Client should be started

Command Steps:

```
elk02:~/working/OS/CLIENT> make
cc client.c -o client -lpthread
elk02:~/working/OS/CLIENT> sh run.sh
Enter any Front end Server IP:
XXX.XXX.XXX.XXX
```