Programming Project 3 of CS5523

Mybooks.com: Extending My Online Book Store

Objectives:

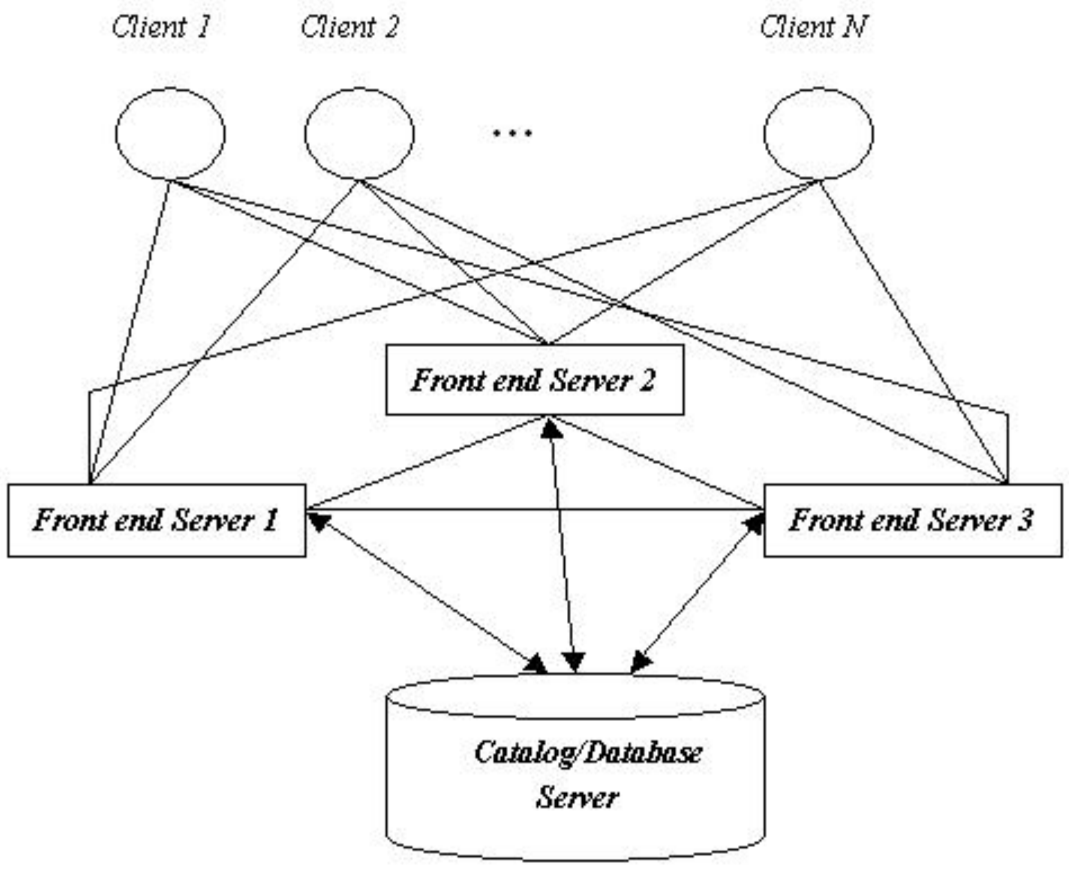
- Familiarize you with two-tiers design;
- Practice different synchronizations in the distributed systems, including Berkeley algorithm or vector clocks.

Project Requirements:

This is an INDIVIDUAL project. You may discuss with your classmates, but you have to write you own code and **copying the code directly will cause you to fail this course**.

Project Descriptions:

(1) Two-Tier design:
Mybooks.com is great success. Thus, you want to extend it since we only have one server in the project1. One basic idea is to add more servers in order to handle more requests. One particular design is to employ a two-tier design: a front-end and a back-end. The front-end tier will accept user requests and perform initial processing. For example, **search()** requests can be answered in the front-end as long as the front-end servers can check with the back-end servers periodically. For the backend, we will only have one server. The **lookup()** and **order()** requests have to be processed by the backend server since the backend server will have the newest information about different books. The figure can be seen in the following figure. For different types of requests, we can refer to the project1 for specific explanations. The number of servers of front-end servers is 3 now, but can be configured to a larger number.



(2) Clock Synchronization.

Mybooks.com is celebrating it's great success since launching it's online bookstore. Thus, it is offering every 100th order a 10% discount for any book of their choice, including 100th, 200th, 300th, etc. Since there are 3 front-end servers, any customer can contact any of three front-end replicas with their requests.

In reality, we may have to choose a timer daemon by a leader election, which will periodically send out synchronization information among all front-end servers. Now let's make this simpler. Assume that we can assign one of three front-end servers in the configuration as the leader. Implement the Berkeley clock synchronization algorithm to synchronize their clocks. Each node then maintains a clock-offset variable that is the amount of time by which the clock must be adjusted, as per the Berkeley algorithm. We will not actually adjust the system clock by this value. Rather to timestamp any request, we simply read the current system time, add this offset to the time, and use this adjusted time to timestamp request.

Every incoming order request at each front-end replica is time-stamped with the synchronized clock value; this time stamp is relayed to the database server for buy requests. The time stamps are used to determine an ordering of buy requests and every 100th buy request is automatically given a 10% discount for the requested book.

Other Notes:

- Initial input: available items for each book can be randomly.
- System timers: you can use gettimeofday() in C/C++ or currentTimeMillies() of System class in Java to acquire the time of serving each request.
- You can use either C/C++ or Java for this project.

Others:

You may choose another algorithm to implement clock synchronizations, such as logical/vector clocks. In this version, we assume that clocks are not synchronized and yet we wish to determine the ordering of events (here events refer to requests made by clients).

To do so, each client process and each front-end server maintain logical clocks. You can either use Lamports clocks or vector clocks for this purpose. Pick either the totally ordered multicast or causally ordered multicast algorithm to determine an ordering of all requests made by the clients to the front-end servers. Note that in this case, requests must be multicast to all processes, in addition to being sent to the front-end server that will process the request. You are free to design the system with either algorithm.

Project Report:

Writing a technical report to describe this project.

Grading Policy:

Code:

- Works correctly: 60%
- In-line documentation and scalable design: 10%
- Testing: 10%

Technical Report: 20%