Distributed Operating Systems

Fall 2020

Lab 2
Turning the Bazar into an Amazon: Replication, Caching and Consistency

---

- This project has the following goals:
  - To teach you Replication, Caching, Consistency
  - To teach concepts of multi-tier web design and micro-services.
  - To teach you basics of containerization and docker (optional).

The Problem:

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

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

These books were added to the catalog during a spring break sale that turned out to be a big success. However, with the growing popularity of the book store, buyers began to complain about the time that the system needs to process their requests. You are tasked with rearchitecting Bazar.com's online store you built in Lab 1 to handle this higher workload.

This project is based on project 1. This assignment is two parts. The first is mandatory and the 2nd is optional.

Replication and Caching

In this part, we will add replication and caching to improve request processing latency. While the front-end server in lab 1 was a very simple component, in this part, we will add two types of functionality to the front-end node. First, we will add an in-memory cache that caches the results of recent requests to the order and catalog servers.

Second, assume that both the order and catalog server are replicated - their code and their database files are replicated on multiple machines (for this part, assume two replicas each for the order and catalog servers). To deal with this replication, the front end node needs to implement a load balancing algorithm that takes each incoming request and sends it to one of the replicas. You may use any load balancing algorithm such as round-robin, least-loaded and can do load balancing on a per-request basis or a per user-session basis. The front end node is NOT replicated. It receives all incoming requests from clients. Upon receiving a request, it first checks the in-memory cache to see if it can be satisfied from the cache. The cache stores results of recently accessed lookup queries (i.e. book id, number of items in stock, and cost) locally. When a new query request comes in, the front end server checks the cache first before it forwards the request to the catalog server. Note that caching is only useful for read requests (queries to catalog); write requests, which are basically orders or update requests, to the catalog must be processed by the order or catalog servers rather than the cache. You can implement the cache server in one of two ways: it can as a separate component from the front-end server and you will then need to use REST calls to get and put items from and to the cache; or your in-memory cache can be integrated into the front-end server process, in which case, internal function calls are used to get and put items into the cache.

Cache consistency needs to be addressed whenever a database entry is updated by buy requests or arrival of new stock of books. To ensure strong consistency guarantees, you should implement a server-push techniques where backend replicas send invalidate requests to the in-memory cache prior to making any writes to their database files. The invalidate request causes the data for that item to be removed from the cache. Feel free to add other caching features such as a limit on the number of items in the cache, which will then need a cache replacement policy such as LRU to replace older items with newer ones.

The replicas should also use an internal protocol to ensure that any writes to their database are also performed at the other replica to keep them in sync with one another.

Like in Lab 1 all components use REST APIs to communicate with one another.

Part II. Dockerize your application (optional).

Bazar.com has decided to use container technology to make it easy to deploy its application on its servers. From our virtualization lectures, we learnt about OS containers and docker. Docker is a tool that enables you to package each application component at a portable docker container. Your goal is to package each of your components: front-end server, cache server (if it is a separate micro-service from the front-end server), catalog and backend server as docker containers.

There are many tools available on the Internet to take the code for each component and package it as a container/ image. You are free to use any such tool to create your docker container images.

Once you have created a docker version of your app, you should also upload it into your project github (in a separate directory), so that you can use docker to directly download it from github and deploy on any machine.

Experimental Evaluation and Measurements

This lab still requires you to do performance measurements or experiments to show/evaluate specific functionality.

1 - Compute the average response time (query/buy) of your new systems. What is the response time with and without caching? How much does caching help?

2 - Construct a simple experiment that issues orders or catalog updates (i.e., database writes) to invalidate the cache and maintain cache consistency. What are the overhead of cache consistency operations? What is the latency of a subsequent request that sees a cache miss?

Put your results in a table and make necessary plots to support your conclusions.

What you will submit

When you have finished implementing the complete assignment as described above, you will submit your solution to github.

1. Source code with inline comments/documentation.
2. A copy of the output generated by running your program. This is similar to lab 1. Be sure to include output for each part of the lab.
3. A seperate document of approximately two to three pages describing the overall program design, a description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). You also need to describe clearly how we can run your program - if we can't run it, we can't verify that it works. **Please submit the design document and the output in the docs directory in your repository.**
4. Performance results of your measurements/experiments should be included in the docs directory. Provide the results/data as a simple graphs or table with brief explanation.