15-440 Project 3: Airbnb with PAXOS

Shaojie Bai (shaojieb), Zirui Wang (ziruiw)

December 11, 2015

Abstract

A project based on 15440 p3 paxos implementation, our Airbnb online booking system is a web-application using PAXOS algorithm for data storage. The interface exposed to the users will be mainly in the form of a web view, while at the back-end server side, we use the classical PAXOS algorithm to keep a distributed replication that ensures data storage consistency.

1 Introduction

Motivated by the need to reach a reservation consensus in most of the online booking system, we worked on this web-application that implements a simple Airbnb website. In the real world, there are many things (steps) involved for a user to make a reservation, including logging into the system, searching for houses (based on criteria), view details, and finally select the housing. Inevitably, in this process there may be lots of problems worth considering. For example, what if two users, at two locations, select the same housing, simultaneously? In our project, we aim to address problems like this that may arise in a multi-user system.

This *naive* version of the Airbnb system supports the basic functionality that a usual reservation service may be able to support. For example, users need to register themselves with a usernmae-password combination, and then log in, so that they can make a booking. Moreover, the user can perform a search operation using the "metric" they specify (e.g.: "New York", "Los Angeles", etc.). Once the search results pop up, the user can have a closer look on the detail information (rating, address, owner, etc.). Finally, if the user is satisfied with a location, he/she can select it by simply clicking a button. Of course, we also let the user to add their own housing to the system and be a host!

2 Design

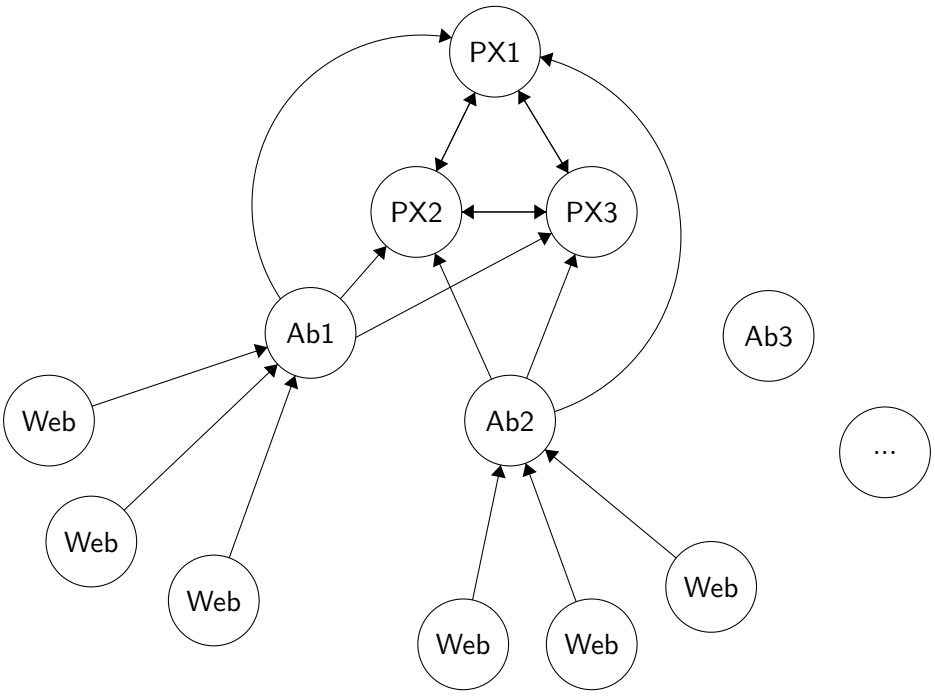
Our design structure can be mainly split into four layers:

- The front-end webview. This is the part that directly interacts with the users. As a result, in this layer, most of the codes were written in HTML, CSS and Javascript.
- The web controllers/handlers. In this part, we implement the handlers in Go. In particular, the handlers deals with the information/requests sent from the front-end webview and serves as a proxy that asks the back-end system the answeres/responses to these questions. Then, once it gets back the result, it renders an html snippet or view so that this information can be displayed on the front-end webview.
- The AirbnbNode. This is what the handlers talk to. When a webpage is served, it opens an AirbnbNode that processes the requests sent from the web. It does not handle the HTML (front-end) side of the work, as controller does; it is simply responsible for managing what to put into the PAXOS and how do we coordinate the reservation requests. The AirbnbNodes act as the last layer of processors before the real PAXOS algorithm: they collect all the information they have, and then propose in PAXOS to reach a consensus in the distributed replication.

- PAXOS algorithm (nodes). This is then the part where we implement the distributed backup. In order to align everything in the backup, we use PAXOS algorithm to reach the consensus. The AirbnbNode, once contacted, will contact a PAXOS node available and get the next proposal number from it by calling `GetNextProposalNumber()`. Then, with the proposal number and the relevant (key, value) pair, the AirbnbNode will ask the PAXOS node it contacted just now to help it propose this request to all the other replicas (PAXOS nodes). Once there is a success, the PAXOS will reply the RPC `Propose()` so that the AirbnbNode is notified and can pass the reply even upward.

Another key part of our design is that, we not only put the housing selection information in our PAXOS. Instead, for everything that needs consensus on, we use PAXOS to seek a back-end consensus. For example, when the user registers itself, we need to pack the new user into a struct and propose that struct (associated with the user) to PAXOS. Once PAXOS accepts this proposal, this means the registratoin is a success. Similarly, for adding housing information and selecting reservation, we also have their own (key-value) pairs that can be used to send to PAXOS.

The structure can be pictured as this. "Ab" stands for AirbnbNode, "PX" stands for PAXOS node.



3 Implementation side

The Airbnb with PAXOS project is a web-application, which means it is based on a standard set of web-developing techniques. For this reason, in this project we used HTML, CSS and Javascript that we wrote to support our front-end interactions with the user. Our css is mainly supported by the Twitter Bootstrap framework v3.3.5, which can be found at http://getbootstrap.com/. It offers a lot of components that are really useful, such as modals, buttons, etc.

The interactive part is achieved mainly using Javascript's jQuery library. These js code changes the html element according to the user's requests, and, in the case that the user wants to talk to the back-end service, it makes an ajax call to the handler layer (who will then forward the call in the form of RPC). A typical ajax we have is like this

Code 1: The ajax call when we get the current reservations of a client user

```
$.ajax({
    url: '/getReservation',
    type: 'POST',
    dataType: 'html',
```

```
    data: { userid: $("#hidden1").text() },
    success: function(data) {
        $("#currentReservationPanel").append(data)
    },
});s
```

For the back-end communication, we mainly use the standard RPC. For example, in the handler, after it dials the AirbnbNode, it simply preprocesses and forwards all the requests to the AirbnbNode. The AirbnbNode will have a collection of the PAXOS nodes it has (it dials everyone when it starts). To contact a PAXOS with arguments in an RPC, the AirbnbNode does something similar to this:

Code 2: The RPC call that the AirbnbNode packs to request permission on a new house

```
proposeNumArges := &paxosrpc.ProposalNumberArgs{newHouse.Owner}
var proposeNumReply paxosrpc.ProposalNumberReply
err := cli.Call("PaxosNode.GetNextProposalNumber", proposeNumArges,
    &proposeNumReply)
if err == nil {
    gob.Register(newHouse)
    proposeArges := &paxosrpc.ProposeArgs{proposeNumReply.N, key, newHouse}
    var proposeReply paxosrpc.ProposeReply
    err2 := cli.Call("PaxosNode.Propose", proposeArges, &proposeReply)
    if err2 == nil {
        reply.Status = airbnbrpc.OK
        return nil
    }
}
```

Notice that we first used an RPC to get the proposal number and then formally propose.

As for the PAXOS nodes, it simply follows the usual Paxos algorithm, which can be found here on Wikipedia. PAXOS nodes, once receive a proposal from AirbnbNode, will send out prepare messages to its peers and act as a leader. Once it gets a majority of advocates, it then enters an Accept phase where it confirms with each other paxos node "Are you sure you are in favor of the proposal with number n_a and value (k_a, v_a)?". Once a majority of other nodes accepts, the commit will follow right away. If the majority is not reached in any step described above, then the proposing simply blocks and waits for up to 15 seconds for a proposal to be reached. The node-to-node communication is implemented using RPC.

Finally, here are the packages/frameworks that we have used in this project:

- Go: `encoding/gob`, `errors`, `net`, `net/http`, `net/rpc`, `strconv`, `sync`, `time`, `flag`, `fmt`, `html/template`, `log`, `regexp`
- Javascript: `Javascript jQuery v1.11.2`, `Bootstrap 3.3.5`

4 How to run?

To run our application, you will first need to download our project from GitHub (to be uploaded later, but it's available on Autolab). Then, you have to put it in your local directory where you put your Go. You will then need to set `GOPATH` to this folder by

```
$ export GOPATH=$‘your_path_of_download\p3’
```

For example, if you downloaded to your local ‘C:\Go’, then `export GOPATH=$’C:\Go\p3’`.

Then, do the following:

```
$ cd $GOPATH/src
```

```
$ go install github.com/cmu440-F15/paxosapp/runners/arunner
...
$ cd $GOPATH/bin
$ ./arunner
```

After the steps above have been completed, you should open another terminal (and similarly, reset your GOPATH). In the new terminal, do:

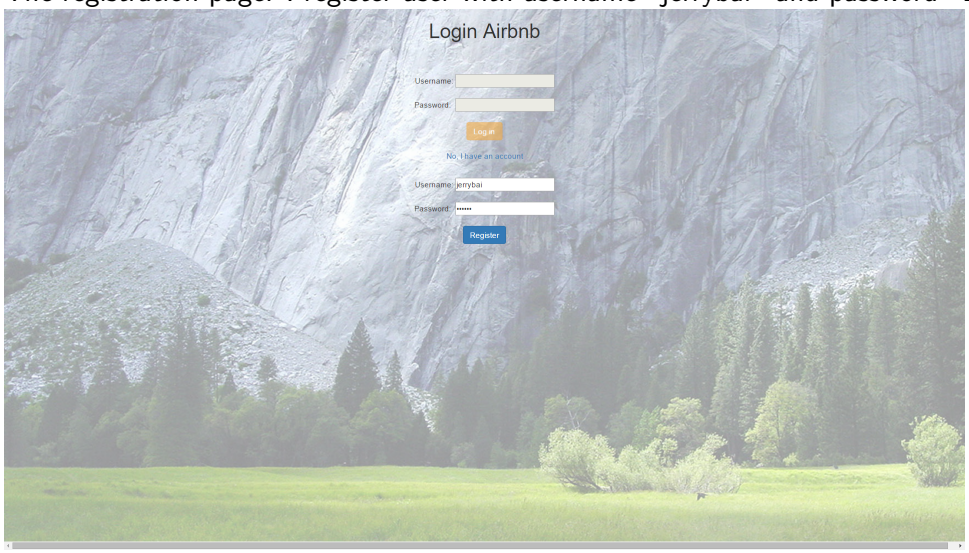
```
$ cd $GOPATH/src/github.com/cmu440-F15/paxosapp/airbnbweb
$ go build wiki.go
$ ./wiki -port=8080
```

Note that we can indeed combine these two terminal windows into one by running `arunner` in the background by `./arunner &`. However, we do this just to give the users a better sense how we organize our work :) Also, in the last step, if you do not specify your own port, it will default to port ":8080".

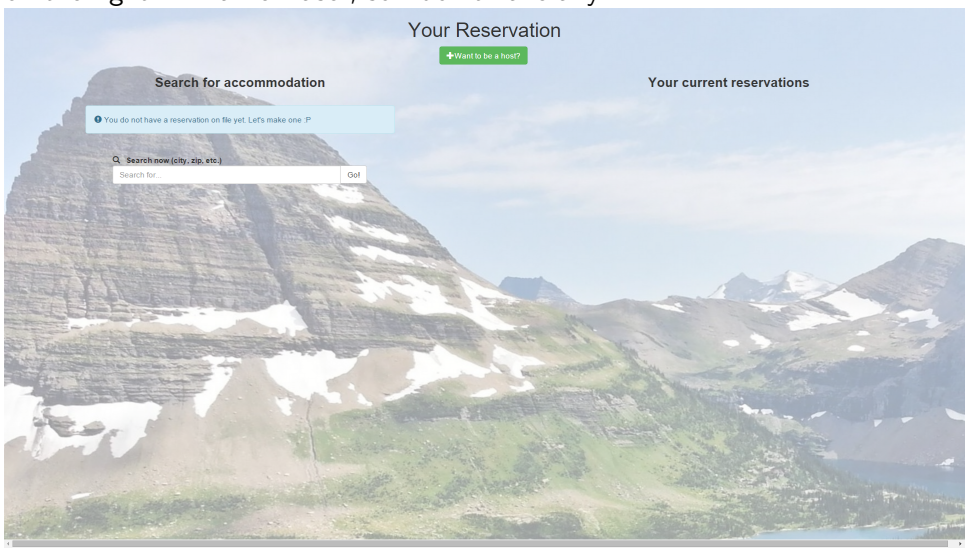
Finally, you can open your browser and type in `localhost:8080/login/`. From there, you can now use the system! For some sample demo, see the next section.

5 Demo

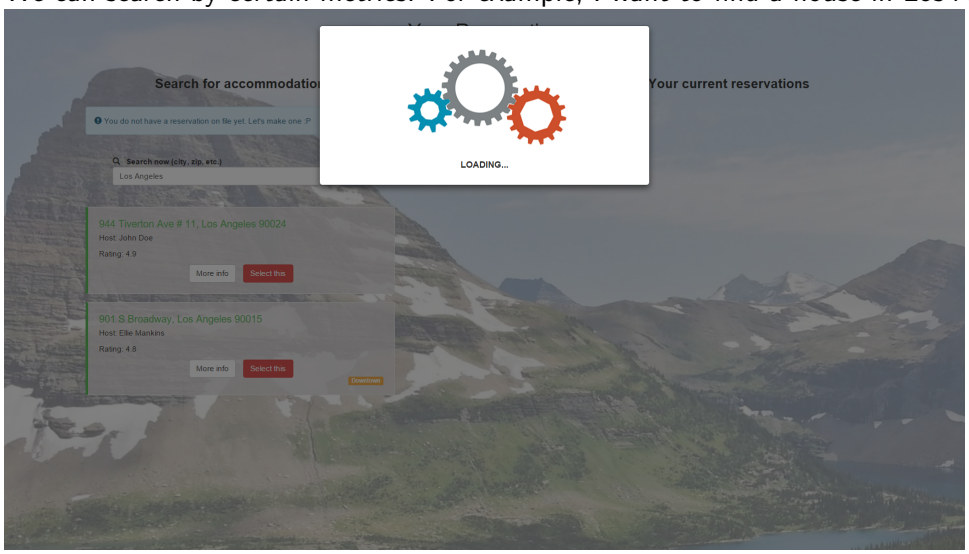
1. The registration page. I register user with username "jerrybai" and password "123123".



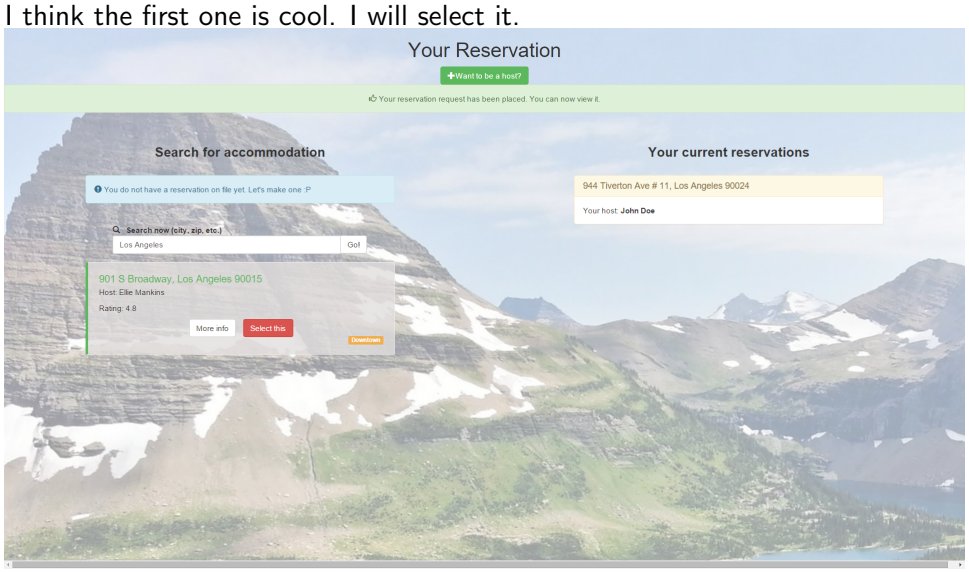
2. After logging in, you will see the reservation page. If you already have reservations, they will show on the right. I'm a new user, so I don't have any.



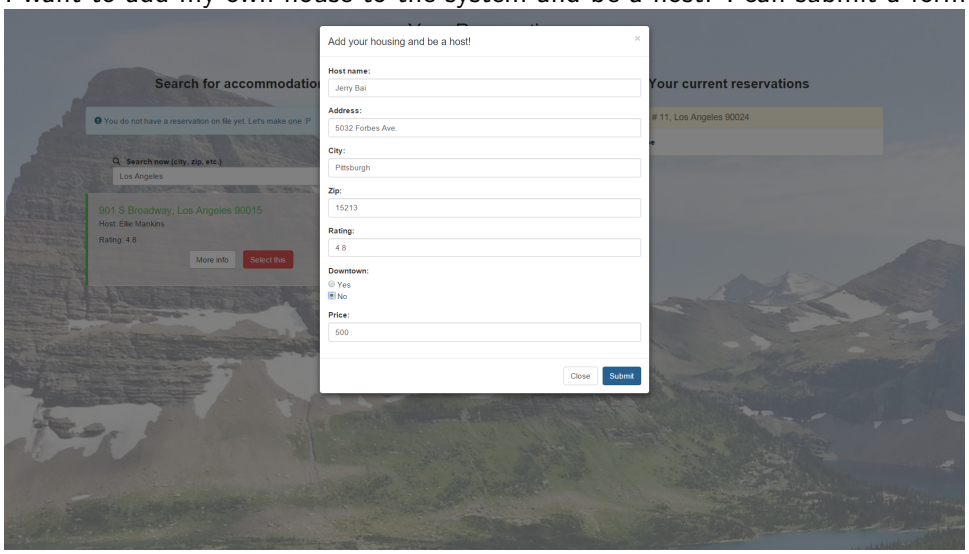
3. We can search by certain metrics. For example, I want to find a house in Los Angeles. Then:



4. I think the first one is cool. I will select it.



5. I want to add my own house to the system and be a host! I can submit a form by



6 Reference

- Bootstrap: http://getbootstrap.com/
- Javascript jQuery: https://jquery.com/
- Golang `http` package: https://golang.org/pkg/net/http/
- Golang "Writing Web Applications": https://golang.org/doc/articles/wiki/
- Paxos: https://en.wikipedia.org/wiki/Paxos_(computer_science)
- CMU 15-440 Distributed Systems: http://www.cs.cmu.edu/~srini/15-440/index.html
- Log in Background Image: http://cdn.bigbangfish.com/beautiful/beautiful-scenes/beautiful-scenes-7.jpg
- Reservation Background Image: http://xdesktopwallpapers.com/wp-content/uploads/2014/01/This-Scene-Can-Only-Be-Find-In-Glacier-National-Park.jpg