15640 Project 3 proposal
Airline Reservation System

Adit Madan
aditm@andrew.cmu.edu

Bo Yang
byang1@andrew.cmu.edu

November 19, 2012

1 Overview

The objective of the project is to implement a distributed airline reservation system. A client of the system can choose to make reservations including flight(s) from one or many airlines. For example, a client may choose to book Airline1:Flight1 from Pittsburgh to New York and Airline2:Flight2 from New York to Boston in a single transaction. To realize this we will build on the key-value store from Project 2 by implementing two-phase commit to support distributed transactions and Paxos for replication/consistency.

A user can enter the system for the following operations:

1. View a list for flights satisfying some criteria. For example: all flights on a specified day from source city to destination city.
2. Make a new reservation.
3. Cancel an existing reservation.
4. View all reservations made by a particular user.
5. Add new flights and/or instances.
6. Delete existing flights and/or instances.
7. Change in pricing or time of flight.

2 System Architecture

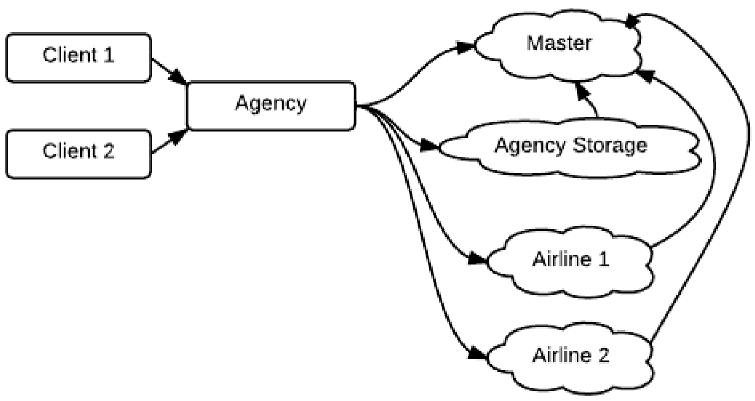

Figure 1: Components of the system.

The system primarily consists of the following components (See Figure 1):

1. **Client:** Initiates one of the requests enumerated in the previous section.
2. **Airline:** An airline consists of a set of storage servers responsible for flight reservations of only that particular airline (such as Southwest Airlines). The airline maintains sufficient replication to be fault tolerant. This cluster is responsible for
   (a) Storing flight information such as departure and arrival time, available slots, price, flight number, airport information and so on.
   (b) Updating flight information such as change of time, list of reserved slots, price and cancellation of flight.
   (c) Providing flight information.

   Each cluster should be able to process multiple requests to update or read flight information at the same time. Therefore, each cluster is composed of multiple servers. For the consistency problem when updating information, we are going to use Paxos. In this way, not only the consistency among all servers is assured, but also the failure of servers can be handled.
3. **Agency Storage** All persistent client information is stored by an agency storage cluster. The cluster of storage servers is responsible of
   (a) Storing user information
   (b) Updating user information
   (c) Providing user information

   Unlike flight information, which is accessed frequently, user information will be accessed much less. Therefore, we would use consistent hashing in each agency cluster. To make sure that the failure of servers would not lead to lost client information, we will replicate each piece of user information on three nodes on consistent hashing.
4. **Agency Server:** All client operations are directed to an agency server. The server is responsible for redirecting communication to one or more airlines and updating the agency storage servers on behalf of clients. For information read requests, application server may simply send read request to servers in corresponding cluster. For basic information updating, which only involves one cluster, application server may simply send update request to cluster. For transactions updating information on multiple clusters, such as travel reservation among multiple airlines, we will implement two-phase commit to assure atomicity of the transaction. If application server doesn't know the address of any cluster, it will ask master cluster, whose addresses are well known.
5. **Master Cluster:** The master cluster is responsible for maintaining addresses of active servers in the system, both agent servers and airline servers. Whenever some server or client needs to send request to another server, it should first get the address of that server. In our implementation, we will keep an IP address list for each cluster. In the list, the master cluster stores the IP addresses of servers which the master server think are in that cluster. When any cluster has new servers or broken servers, it should tell master cluster the new information.

   Since the master cluster also needs to handle multiple requests simultaneously and servers in it may also fail, we will use Paxos for this cluster as well.

3 Interface

Different components of the system interact with each other via RPC calls. Following is an outline of the basic function calls:

```
A. Client => Agency Server

// Calls made by end user
CreateUser(userid string) status Status
GetFlights(departure Date, arrival Date) (flights []FlightInfo, status Status)
BookFlights(flights []FlightInfo) status Status
CheckReservation(userid string) (fights []FlightInfo, status Status)

// Calls made my airline admins
CreateFlight(flight FlightInfo) status Status

B. Agency Server => Airline
GetFlights(departure Date, arrival Date) (flights []FlightInfo, status Status)
BookFlight(flight FlightInfo) status Status

C. Agency Server => Agency Storage Server
// Store/Retrieve information about reservations made
GetUserInfo(userid string) (userinfo UserInfo, status Status)
StoreBookingInfo(bookinginfo BookingInfo) status Status

D. Agency Server => Master
// Get list of IP addresses
GetAgencyStorageServers(agencyid string) (addresses []string, status Status)
GetAirlineStorageServers(airlineid string)  (addresses []string, status Status)

E. Airline/Agency Storage Servers => Master
// Update master in case new servers are added/removed from the cluster
UpdateAirlineStorageServers(airlineid string, addresses []string) status Status
UpdateAgencyStorageServers(agencyd string, addresses []string) status Status
```

4 Test Plan

4.1 Correctness Test

1. **Airline Storage Cluster**
   (a) Store and fetch information on the cluster to/from different servers in different permutations in order to make sure that information stored on different servers of the same cluster is consistent
2. **Agency Storage Cluster**
   (a) Servers should be able to correctly calculate the position of a key on the hash ring
   (b) If the key of a request is wrong, server should be able to return the current servers information
   (c) Duplicate copies of server information on the hash ring should be consistent
   (d) Servers should be able to update the server information if it get error report from application server
3. **Master Cluster**
   (a) Store and fetch information on the cluster to/from different servers in different permutations in order to make sure that information stored on different servers of the same cluster is consistent
4. **Agency Server**
   (a) Should be able to correctly send request messages to airline, agency and master clusters
   (b) Should be able to randomly choose IP address of a cluster from the address list given by the master cluster

(c) Should be able to send an error report to the next server in the address list if previous servers are unable to connect

(d) Two-phase commit should abort in case some storage server is not able to process request

4.2 Robustness Test

1. **Airline Cluster, Agency Storage Cluster, Master Cluster**

   (a) Remove several servers (randomly) to see whether the cluster can work correctly

   (b) Add several new servers to see whether the cluster can work correctly

4.3 Performance Test

1. **Airline Cluster, Agency Storage Cluster, Master Cluster**

   (a) How many read and update requests can the cluster handle? Make a large number of s-tore/retrieve requests and validate fetched results.

2. **Agency Server**

   (a) How many client requests can it handle? Make a large number of conflicting reservation requests to a single agency and check if the system makes progress.

3. **Whole System**

   (a) Make a large number of conflicting reservation requests to multiple agency servers and check if the system makes progress.

5 Tiers

5.1 Tier 1: Must have features

1. Implement all application logic like view flight information, make reservation and cancel reservation in Client + Agency Server against a system which has a single instance of Airline/Master/Agency Storage clusters with a single node each.

2. Add multiple airlines, with a single storage server each, to the system and implement two-phase commit.

3. Support multiple agency storage servers to distribute client information (consistent hashing).

5.2 Tier 2: Make the system robust

1. Implement Paxos and deploy on airline storage clusters containing multiple nodes.

2. Similarly create consistent replicas for master cluster.

5.3 Tier 3: Nice to have features

1. Add application level features like a flight timing change

2. Create a GUI for the client.

3. Implement on Web Server

6 Schedule

1. Tier1: Nov. 20 to Nov.25

2. Tier2: Nov. 26 to Nov.30

3. Tier3: Dec. 1 to Dec.4