Project Proposal

Chun Chen chunc@andrew.cmu.edu
Bo Ma bom@andrew.cmu.edu

Description of the application

Our team plans to build an airline reservation system. Our system supports ticket booking and ticket checking. In the system the users can check the ticket status of a specific flight from the airline, and also reserve the ticket that they want. After the user made the reservation, they can check their tickets anytime they want. On the other hand, we have airline administrators who manage the airline tickets. Airline administrators can change the time and the ticket status of a flight anytime they want. Airline administrators can also add new flights.

Overall structure

The overall structure of our reservation system is as figure 1.

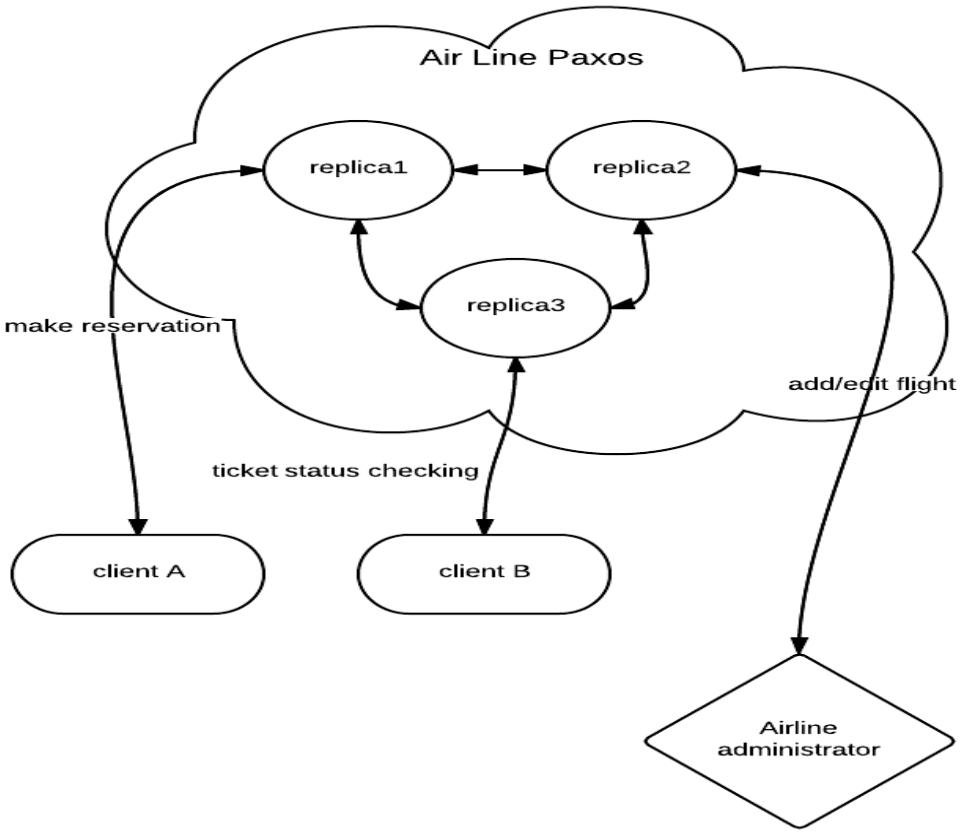

Figure 1. The overall structure

Our system consists three major components:

Airline Paxos: As shown in the figure above, the big circular represents the airline replica servers. They use PAXOS protocol to communicate with each other. so when one of the servers receives the client's request which wants to book the ticket. the airline replica servers should select the leader and get the replies from majority of server to commit. Maybe there are many processes believing they are leaders, but the protocol only guarantees progress if one of them is eventually chosen. We support failure tolerance for the replicas. If less than half of the replicas fail, our airline system can still works.

Client: In the figure, we have two client, clientA communicates with replica1 to make the reservation. clientB communicates with replica2 to check the ticket status. Several clients will be allowed to communicate with different replica server at the same time. Thus, the Airline paxos server should handle the situation like this. If the replica1 is selected as the leader first, it send the message to the other replica and after it gets the reply from majority of replicas it should commit and make reservation. Then replica2 is selected as the leader. Replica2 should first update the status from replica1 and then executes the ticketing checking operation for clientB.

Airline administrator: Diamond in the figure means the Airline administrator. The Airline administrator can manage the airlines. It will communicate with Airline replicas to add/edit flight that user can book.

Distributed features and algorithms

The major distributed feature in our system is PAXOS.

There are multiple replicated servers inside the airline. We plan to build a PAXOS instance on top of these replicated servers, so they can keep consistent and tolerating losing up to half of the nodes. Here consistency means all replicated servers will execute the same sequence of operations in the same order, which is guaranteed by PAXOS; as for the fault tolerance, it refers to the system can still keep working if no more than half of its nodes die.

As for more details about our system, we plan to "PAXOS" the operations for every time slot (in this project, checking tickets, making reservation for a specific flight are all operations in PAXOS). In this way, all nodes will get consensus on the operation for every time slot, which guarantees the consistency and fault tolerance, as discussed below:

1. Consistency
Consistency basically means all replicated servers should execute the same sequence of operations in the same order; in another word, when a client sends a request to a replicated server, we should guarantee the server already gets the latest status that all nodes have

consensus on before executing the new coming operation. This can be achieved by the workflow shown below:

Step 1. Client sends an operation |op1| to a replicated server
(Assume the replicated server has already executed |Nop| operations before)
Step 2. The replicated server asks PAXOS: “I want to store |op| in time slot |Nop| + 1”
Step 3. PAXOS executes and answers: “time slot |Nop| +1 has a consensus on |op2|”
Step 4. Update the local states of the replicated server using |op2|
Step 5. If |op1| and |op2| are two different operations (we can assign each operation an unique operation id for comparison), increase |Nop| by 1 and goto step 2; otherwise we are done.

We can see in this way, all replicated servers are guaranteed to execute the same sequence of operations in the same order.

2. Fault Tolerance
PAXOS guarantees the system with ‘2f +1’ nodes will still work even if ‘f’ replicated servers fail. It also guarantees that if a replicated server goes down and get recovered after a period of time, it can still work fine cooperating with other servers.
From the workflow shown in the Consistency section above, we can see that if a replicated server recovers with all its state lost, it can still catch up to the latest state that all nodes have consensus on by PAXOS (It will run PAXOS for all time slots and execute the operation that all node have consensus on for every time slot, thus recover to the latest and consistent state)

Testing plan

We are planning to test on four aspects as below: (we will pay more attention on the Reliability and Recovery part)

- Functionality: We will test the normal flight queries from the client side and the administrator side, including checking flight remaining tickets, making flight reservation, checking reserved flight from the client side; and adding new flight, changing flight’s remaining tickets from the administrator side.

- Reliability: We will randomly kill less than half of the nodes, and check whether the system still works properly.

- Recovery: We will kill some nodes and bring them back after a period of time, wait until recovery, and then kill some other nodes, then check if the system works properly.

- Deadlock free: We will test our code carefully to prevent deadlock.

Expected tiers

Basic Tier	1. Implement the basic PAXOS algorithm for replicated servers of the airline. In the basic tier we assume the replicated servers of the airline are static, and each client will have a hostport list of all replicated servers before contacting with the airline. 2. Implement a basic command line UI and the users can use it to make ticket queries. 3. All nodes are represented as processes on computer.
Advanced Tier	1. Deploy nodes on different machines rather than the several processes on the only one machine. This will strengthen the reliability and scalability of the system. 2. In the basic tier we assume the replicated servers of the airline are static. However in the advance tier we plan to provide a tool to add/delete replicated server for the whole system, and the clients can get the updated list of replicated servers dynamically. In this way the administrator of the replica cluster can manually add/delete nodes when needed.

Detailed plan of implementation

4.9 - 4.16	Write project proposal.
4.16 - 4.23	Implement the PAXOS algorithm.
4.24 - 4.25	Finish the client, administrator and UI side work.
4.26 - 4.28	Finish the test work and prepare for presentation.