Fall17 CS271 Project3

November 7, 2017

UCSB is famous for its soccer games. There are several ticket offices (assume there are 5) located around the campus. In previous years, if we had 100 tickets, each kiosk would get 20 physical tickets and sell them. People may wait in lines for an hour just to know that this kiosk is out of tickets while other kiosk might still have a plenty of tickets left. This year the ticket office decided to employ you as a software engineer to help them improve this process using your distributed systems domain knowledge.

Each kiosk will have a ticket printer, so if one kiosk can make sure that the ticket pool has enough tickets and it's the only kiosk that is accessing the pool, it can print the number of tickets and sell them to the customer. The problem looks familiar, right? We can have a simple solution use Lamport Mutual Exclusion. However, the campus network has been suffering from instability. Sometimes a kiosk cannot connect to the network and the whole ticket selling system stops working since the requests cannot get enough replies. Moreover, a process may crash, and a kiosk can fail. After a failed kiosk restarts, it cannot remembered what happened before! These can not be handled by Lamport Mutual Exclusion.

But hey, we have just learned Paxos, which is a great approach for solving this kind of site failure, and network partitioning problems. Let's design a system using Multi-Paxos.

1 Data Management Protocol

Multi-Paxos ensures that there is a totally ordered log. Hence all kiosk logs will have the same order. It will also handle failures. In particular, if a kiosk fails, the ticket selling history and the number of tickets in pool will still be available after restarted, the system can still sell tickets as long as a majority of servers is available. As Lamport's paper suggested, we should elect one of the kiosk as a leader. When leader is stable, the system does not need to go through "Prepare Phase". The leader can just run "Accept Phase" when requests come. For tackling the leader failure or network partitioning (use heartbeat to confirm whether the lead is still alive), the system should be able to select a new leader.

Additionally, to ensure the smooth decommissioning of kiosks, the system will have the ability to cope with Configuration Changes, using Multi-Paxos's configuration change mechanism.

2 Application Protocol

In this project, it is not required to ensure causality between requests, as long as, the MUTEX problem is solved.

3 Implementation Detail Suggestions

Before you start, you should have a good understanding of Multi-Paxos, so that you can design your data structures and program behavior effectively. Try to implement the Data Management Protocol (Multi-Paxos) first, ie, focus on the Multi-Paxos implementation, and do not worry about the contents of the log. You can simply create a stub log type with nothing in it at the beginning. Once you can ensure that the log can be replicated, site failures can be handled etc, you can then continue and add the application logic to solve the ticket selling problem.

4 User Interface

We will have 3 datacenters up at the beginning. And the configuration change will add two more datacenters to the cluster. Each datacenter should have a client connected to it and receive commands from the client.

Client Commands:

1. buy numOfTickets
   Explanation: Buy numOfTickets tickets. And we should be able to see the result from the client interface after the request has been processed.

2. show
   Explanation: This command should (1) In the first line, show the state of the state machine for the application (2) In the following lines, show the committed logs in the datacenter the client connected to.
   item change param1, param2 ...
   Explanation: Configuration change command. You should provide the parameters depending on what information you need in your configuration change logic.

Datacenter Logging:
You should carefully think about what messages should be displayed at the datacenter. We should be able to see the logs for each command during the

demo and should be able to infer what the implementation is doing from those logs. E.g Do not log every heartbeat message during the demo. It can be useful during project debugging phase but will make it impossible to decipher what is happening during demo time.

5 Grading Scheme

As this part of the project has multiple tasks, the grading will be based on the completed tasks as follows.

1. Normal operations and failure of a non-leader node: 35%
2. Tackling Leader Failures: 30%
3. Tackling Configuration Changes: 20%
4. Solve the Ticket Salex problem using Multi-Paxos and log: 15%

6 Deadlines, Extension and Deployment

This project will be due 12/7/2017. We will have a short demo for each project on Friday 12/8/2017. For this project's demo, you can deploy your code on several machines. In particular, we would like you to either use an AWS setup or a Eucalyptus environment on campus.

7 Teams

Projects should be done in team of at most 2. You can use Piazza to form teams.