Consensus Programming Project

CSCI 520: Consensus Programming Project

Due: 4/1/18

Introduction

In this assignment you will continue to hone your distributed systems development skills. You will implement a distributed consensus protocol and use it as a building block for a distributed application.

Assignment

Your task is to implement the Raft distributed consensus protocol and then use it to implement a distributed version of Rock 'Em Sock 'Em robots. Note that although there are many Raft implementation available on the internet, you will need to implement yours from scratch - undue similarity to publicly available code will be seen as plagiarism.

Rock 'Em Sock 'Em Robots

From Wikipedia we learn: "Rock 'Em Sock 'Em Robots is a two-player action toy and game. It features two dueling robot boxers, Red Rocker and Blue Bomber, mechanically manipulated by the players, and the game is won when one player knocks the head off of the opponent." You can see a lifesize version the game in action here!

Your distributed program should allow human players on two different computers as either Red Rocker and Blue Bomber. Each robot can be directed to perform the following actions (slightly modified from the original game):

- `punch_with_left()` [Q]
- `punch_with_right()` [W]
- `block_with_left()` [A]
- `block_with_right()` [S]

When a robot blocks, it enters a `blocking_with_left`, or `blocking_with_right` state, which persists until another action is taken. When `blocking_with_left` opponent's punches with the right are blocked. Analogously for `blocking_with_right`, punches with opponent's left are blocked.

When punching an opponent who is not blocking, a robot has 10% chance of landing a hit. If a hit lands, the opponent's head is knocked off and the the match is over. A robot can only punch once a second. After being blocked, a robot cannot punch for three seconds with either hand.

One of your tasks will be to define a user interface that lets a user take actions and receive feedback as

to its effect. Because your system includes two players, you may need to write scripts that trigger their actions.

Raft

You will host your game on using two clients and five servers. We will assume that clients do not fail, but the servers and communication between them can. The role of the servers is to maintain game state and arbiter the order of player actions.

The five servers will run the Raft consensus protocol to maintain between them log of player actions. To implement the game, the servers will also need to use this log to maintain game state, specifically whether a player is currently blocking and whether they are allowed yet to dispatch another punch.

In addition to normal Raft operation, your servers need support the following operations:

- `fail()` - the server 'crashes' and loses all local state
- `recover()` - the server 'recovers,' rejoins the Raft protocol, and recovers the log and state
- `timeout()` - the server decides that it has not heard from the coordinator within a timeout

You will use these operation to demonstrate the robustness of your Raft implementation.

Deployment

You must implement a system with five server nodes. Each node will be run on a different machine. The log should be written to permanent storage, so that it will survive node crashes. You need to provide, at least, a minimal user interface (UI) for players to take actions. A text-based UI is fine. The contents of the log must also be viewable, for example by reading a file, so long as the file is human readable.

You will deploy your project on Amazon EC2, on five machines in different regions. You will use micro-instances. You should sign up for AWS Educate to get your $35 credit for use of Amazon's cloud infrastructure. This should be enough for you to deploy and demo the project. You should not do your development in AWS, but push your code from a repository (GitHub, BitBucket) to your AWS instances to test and demo. Please be careful to use a private repository, or otherwise protect private keys so that your AWS accounts are not hacked.

Grading

Your grade will be based on a screencast that demonstrates the functionality of your code. I expect the demonstration to show the different types of events that players may take. You also need to demonstrate the correct operation of your Raft implementation through server crashes and timeouts. I will be looking for functionality and correctness of your mechanisms.

Specifically, you will be graded according to the following rubric:

12 points - Raft
2 - nodes can elect a leader after system start
1 - nodes can re-elect a leader after leader fail
1 - node can become leader after it times out

2 - leader can add events to its log
1 - log event structure is correct
2 - leader replicates log
2 - events in log are correctly marked as committed
1 - a node can recover its log after it fails and misses some events

6 points - Rock 'Em Sock 'Em
1 - robots can punch - action written onto the log
2 - robots can block - action written onto the log and removed after another action
3 - robots head knocked off after successful hit - action taken correctly (describe how your program makes the decision)

2 points - AWS
1 - implemented on different AWS nodes (or another cloud service)
1 - nodes use permanent storage

What to submit?

1. A link to a YouTube video showing your code running on AWS instances.

2. Your source code (unzipped) for D2L plagiarism check

Published by Google Drive – Report Abuse – Updated automatically every 5 minutes