Project 3 Proposal: Go Plays Go!

Won Gu Kang (wonguk)
Dave Best (dbest)

Application Description

The application will basically be a server that runs user submitted AI programs for the game Go, hence the name “Go Plays Go!”. The application as a whole should basically try running each AI with each other and rank the AIs given their wins and losses. The purpose of this is to streamline testing a large variety of A.I’s which would typically be a very heavy task (in terms of sheer amount) to do linearly on one machine but when spread out across multiple machines and process can greatly reduce the time needed along with approving scalability.

Overall Structure

Main Server

The main server will be responsible for receiving AI code over a period of time and at some point closes and runs them on worker servers to be evaluated. The main server will then send the different AI submissions to the different worker servers and keep track of the results for each game between the AI programs. Since the main server keeps track of all this data, and is basically the master in the system, we will replicate it by using paxos.

Worker Server

The worker servers will be responsible for receiving the AI code from the main server and letting the two AIs play with each other. The worker will setup the environment (Board and “communication” between the two A.I code) for the pieces of A.I determined by the main server. All of this will run in go routines on the workers as a go routine for each A.I and a go routine for the board**.** After the game ends, we will send results of the game back to the master server.

Distributed System Features

RPC

The different servers within the system will communicate with each other through RPC calls of functions that will be predefined in the system. This will allow us to handle errors more easily, compared to using other methods of communications. Although we haven’t actually started defining the RPC functions, we will basically have functions to send AI programs to the main server, to send the AI programs to the worker servers to play the game with each other, and to send the main server back the results of the game that was played. We will probably add more

functions to access the data within the Main server, such as the different wins and losses for the different AIs, which we could possibly link to a Web GUI later on in the project.

Paxos

Because the main server hold all the AI programs and all the results from the worker server, we plan to add robustness to the system by replicating our main server with paxos. This will allow our system to be running if a given main server goes down.

We decided that replicating the worker servers was not as crucial because if a worker goes down, we can simply re distribute the tasks that particular worker was working on to other workers.

Testing

There will be basic unit tests to check that the application follows the rules of Go. There will be more unit tests for the master and slave servers to check that they make the right RPC calls to each other given certain circumstances.

When most of the application is developed, we will implement more acceptance tests for the system as a whole so that we can test out the interactions between the servers. The acceptance tests will be in the form of shell scripts, as done in the previous homework assignments.

More overall system tests will include shutting down certain servers to check the robustness of our paxos implementations. Finally, we will add tests for 2PC where some worker servers refuse certain jobs.

System Implementation Tiers

1. Define basic RPC functions for the different servers that should be implemented for the master and slave servers
2. Implement the system without Paxos or 2PC to get the basic framework working. Also implement the “referee” for the game of Go
3. Add basic unit tests to test out the Go rules, and basic acceptance tests for the system to make sure the basic functionality of the system is held
4. Implement Paxos and 2PC within the system to add robustness
5. Add more acceptance tests to check out the robustness of the system.
6. Implement simple AI go programs to test out the system
7. Possibly add a Web GUI to monitor the progress of the system and submit AI code.
8. Modify the system to possibly allow a human player to play with (change to watch as playing could really slow down the system) an AI in the system

Schedule

April 16 - 17:

Define the different RPC functions that will be needed within the system, and other interfaces, such as the interface for the go AI code, so that we can start working in parallel on the different parts of the system.

April 18 - 21:

Try to complete the system up to Step 3, described above, where we have the basic functionality of the system working, with tests to check their correctness. Also have the different runners for the clients(code submitters), master servers, and slave servers ready to demonstrate the functionality of the system.

It would be better to have some simple Paxos or 2PC features implemented before the first code review.

April 22/23:

Do Initial Code review with the course staff to get some feedback on the system and report our progress.

April 22-25:

Implement Paxos and 2PC within the system so that we add more robustness to the system. We will try debugging by running the system to the tests we've already defined, and then adding more tests to test out the robustness of the system (such as losing a master server in the paxos group).

After the system is stable with Paxos and 2PC, we will try to make the system handle errors that can arise better.

April 26-29:

Here, if we are on schedule, we will try to add a web GUI for our system so that we can have a visualization of the system running. The web GUI will possibly be a separate from the initial system we have in hand, just so that we can keep our code modular. We could possibly integrate the web GUI into our master servers in paxos, but that may cause certain problems here and there.

Also, if the monitoring web GUI is done and working, we will try to implement another simple web GUI where we allow a human play with any AI that is within the system. Hopefully this

is implemented, so that during the code reviews and demonstrations, people will have a better feel of our system.

April 29/30:

Final code review and demonstration with the course staff.

May 1:

Final submission and presentation.