Project 3: Design Your Own Distributed System

1 Overview

In this project, you will design, implement, and thoroughly test a distributed system by implementing your own application, such as a multi-player game, a collaboration tool, or a transaction system.

You *must* work on this project **with a partner who is also in the class**. However, before you make any decisions, please refer to the late day policy to avoid any last-minute surprises regarding partner-projects and late days.

2 Project Requirements

Since this is a course in distributed systems, we want your project to have "interesting" features from a systems perspective. Here are some important properties your system should have:

- Your system must involve either **Paxos** or **Two-Phase Commit** in some way, shape, or form.
- The system must support multiple, autonomous agents (either human or automated) contending for shared resources and performing real-time updates to some form of shared state.
- The state of the system should be distributed across multiple client or server nodes. The only centralized service should be one that supports users logging on, adding or removing clients or servers, and other housekeeping tasks.
- The system should be robust. It should be able to continue operation even if one of the participant nodes crashes, and it should be able to recover the state of a node following a crash so that it can resume operation.
- If you have something you are really, really dying to do that does not involve Paxos or 2PC but believe it satisfies the spirit of the 440 project goals, you must have written permission from the instructors. Talk to us first.

We will let you choose your own application, and we will give you wide latitude in the overall design of your implementation. However, keep in mind the following requirements regarding your implementation:

- You may use any programming language you wish.
- Since it is difficult to predict just how hard implementing a new system will be, you should formulate as a set of "tiers," where the basic tier is something you're sure you can complete, and the additional tiers add more features, at both the application and the system level. For example, your system enhancements could include increasing the level of fault tolerance, decreasing the time to recovery, or adding features to increase scalability or security.
- You may use external libraries if you wish (with the exception of advanced distributed system packages, such as libraries which implement the Paxos or Two-Phase Commit protocol for you). You may also use external packages and SDKs to support the application or user interface (i.e., GUI toolkits/frameworks, image/video/audio processing packages, online mapping APIs, etc.).

3 Evaluation

Your system will be judged mainly on how it operates as a distributed system. The primary evaluation will be according to whether your system has the following attributes:

- It should be an interesting distributed system, making use of the Paxos or Two-Phase Commit protocols (and optionally other algorithms we have covered in class as well, such as ones for distributed synchronization, replication, fault tolerance and recovery, security, etc.).
- The software should be well designed and well implemented, in terms of the overall architecture and the detailed realization.
- You should devise and apply systematic testing procedures, at both the unit and systems levels.
- The system should operate reliably and with good performance, particularly in the face of failures. You must have at least one test case that *demonstrates* this resilience to failures that you can show off in the project review with the course staff.

Important, but secondary considerations include:

- How interesting is the application? Is it a fun game, a useful tool, or a solution to a real-world problem?

- How nice is the application's appearance: does it have a nice interface or a compelling visual display?

When we grade your project, we will expect to see your source code, see your project run, and see your tests run.

4 Project Ideas

The following list of projects is intended to spark your imagination. Feel free to come up with something totally different. Contact one of the teaching staff if you are uncertain of the suitability of your ideas.

- Projects with a real-time multi-user component are a good match for Paxos —you could either have the individual users be part of the Paxos quorum, or use Paxos for the replicated state that stores their updates.
- Shared document editing, in the style of Google docs. The system should support real-time editing and viewing by multiple participants. Multiple replicas would be maintained for fault tolerance. Caching and/or copy migration would be useful to minimize application response time.
- A simulated life game, in the style of The Sims or Farmville. 'The state of the system would be partitioned spatially, with replication for fault tolerance.
- A multi-player real-time game—there are many possibilities here.
- An airline reservation system. Each airline would maintain its own collection of servers, with enough state replication to enable automatic fail-over. It would be possible to book travel that involves multiple airlines.
- A low-latency notification system. E.g., watch a whole bunch of RSS feeds and send all subscribers an email when one is updated. Interface with both the raw RSS feeds and Google's update notification service. Replicate and partition the state of the monitoring system so that it can scale and survive node failures.
- Projects involving multiple agents or users coordinating updates that can be shown on a map (e.g., Google Maps). Integrate this with Google Maps and some AJAX—makes for a good easy GUI.
- Ambitious projects in the past have involved creating Android/iOS smart phone apps and games.

- Implement a distributed filesystem that does something interesting. Maybe you want one for storing your MP3s or movies. Or perhaps for something entirely different. What design constraints and optimization criteria would you have?
- A solution to a problem that really bugs you or that you would like to see solved—perhaps your new startup idea!

5 Project Deadlines

This project will have four deadlines, each discussed below:

5.1 Project proposal (due Tuesday, April 15 at 11:59pm)

Before you submit your final project, you must submit a project proposal consisting of the following:

- A description of the application.
- The overall structure of the implementation.
- The distributed system features and algorithm(s) you intend to implement.
- Your plan for testing the system.
- How you intend to implement your system as a series of tiers.
- A schedule for how you plan to carry our your design and implementation.

This document should be around three to four pages long, and it will count for 25% of the total grade. We encourage you to meet with one of the course staff during the planning stages to get some feedback on your ideas prior to submitting your proposal.

To submit your proposal, you should hand-in a document named `proposal.pdf` to Autolab by the **Tuesday, April 15 at 11:59pm**.

5.2 Preliminary demonstration and code review (Tue-Wed, April 23-24)

During the week after you submit your project proposal, we will schedule 30-minute time slots where you will meet with one of the course staff. You should be able to give a demonstration where some aspects of the system are working. You will go over your code,

and you will discuss how you will complete the project. This review will count for 15% of the overall grade.

5.3 Final demonstration and code review (Tue-Wed, April 29-30)

After your first meeting with the course staff, we will also schedule a second and final 30-minute time meeting. You will give a final demonstration and a code walk-through, including how you did your testing. This will count for 60% of the overall grade.

5.4 Final submission & presentations (due Thursday, May 1 at 10:29am)

We will have a final demonstration day in class on **Thursday, May 1 at 10:29am**, where we will let groups (voluntarily) present about their work and let others try out the applications and learn about their implementations. The projects voted best by their classmates will receive prizes to be announced.

To submit your final project, you should hand-in a `tar` to Autolab by **Thursday, May 1 at 10:29am**. The file should be named `p3.tar` and should include all of the source code you wrote for the project, as well as a `README.txt` file explaining how to run the project, run any tests you wrote, etc.