CPSC 551 - Distributed Systems - Fall 2019

Project 3, due December 16

Background

In Project 1, you started with some relatively simple constructs: synchronous Remote Procedure Call, the Adapter pattern for middleware, and Linda tuple spaces.

In Project 2, you used microblogging as as one particular application of tuple spaces to explore a variety of distributed systems concepts, including multicast, logging, recovery, naming, message delivery, and replication. Along the way, you should have encountered and documented a variety of errors and unexpected behaviors. While some of these may have been due to bugs in the code, several are due to flaws in the design.

In this project, you will rectify some of the problems you encountered in Project 2 using concepts and algorithms from Chapters 6 (Coordination), 7 (Consistency and replication), and 8 (Fault tolerance) of the textbook.

System model

The basic model for this project remains the same as Project 2: each user has a Rinda tuplespace. Each user reads and writes microblog messages only by interacting with their own tuplespace, but all messages should be visible to each user as if they were connected to a single centralized server. The contents of a tuplespace should survive server restarts.

Note that this description reflects the *model*, not the architecture. In the previous project you explored several designs for system components to implement this model. You may take any approach to software or system architecture in this project, whether it involves modifying components from previous approaches in Project 2 or entirely new designs for this project.

Code, servers, and modifications

Restrictions from previous projects (e.g., confining solutions to basic tuplespace `._in()`, `._out()`, and `._rd()` operations; or using existing server functionality as-is without modification; and even which functionality is implemented by which component) are lifted.

You may re-use or modify any code from Project 1 or 2, including your own code and any code provided. You may program in either Python 3 or Ruby as appropriate, add additional libraries such as ZeroMQ, and modify the behavior of servers and clients as necessary. You may remove existing system components and define new components and services as long as the underlying model is preserved.

Identifying problems and implementing solutions

In Project 2 you should have briefly outlined a number of different problems or unexpected behaviors that you encountered with the various approaches taken to messaging and delivery or with the design of various components and their interactions.

In this project, choose two of these problems to focus on. Document in detail

- What the problem is
- What triggers the problem, and how to reproduce it
- What impact the problem has on the system
- What the correct behavior should have been

For each problem you choose, select an approach to addressing it based on the algorithms and techniques documented in Chapters 6-8 of the book. Examples include (but are *not* limited to):

- Total-ordered multicast
- Election and voting
- Client-based (pull) protocols
- Local-write protocols
- Consensus algorithms
- Failure detection

Document your approach and demonstrate that your implementation solves the problem you identified.

Note that you need only solve the problems that you have identified; other problems with the system may persist. For example, if you focus on issues with reliable replication, the contents of a tuplespace may still fail to survive a restart (though it might ask for missed updates later). Or if a server failure is detected and recovered from reliably, it may still have missed updates while it was down.

Platform

You may use any platform to develop and test the project, but note that per the Syllabus the test environment for projects in this course is a Tuffix 2019 Edition r2 Virtual Machine with Python 3.6.8 and Ruby 2.5.1p57. It is your team's responsibility to ensure that your code runs on this platform.

Submission

Submit your project by uploading your Python and Ruby source code, documentations, and any other relevant artifacts to the `project3/` subdirectory of the folder that was shared with you on Dropbox.

You may work alone, or make a single submission for a team of 2-3 students. If you work in a team, only one submission is required, but for safety consider uploading copies to each team member's submission folder. (Make certain, however, that the copies are the same in each case; the instructor will not attempt to ascertain which is the “real” submission.)

A printed submission sheet will be provided on the due date. To finalize your submission, fill out the sheet with the requested information and hand it in to the professor by the end of class. Failure to follow any submission instructions exactly will incur a **10%** penalty on the project grade for all team members.