CPSC 551 - Distributed Systems - Fall 2019

Project 2, due November 18

Background

One of the examples of shared tuplespace usage from the previous project was microblogging, with users alice, bob, and chuck posting and reading messages.

For this project we will continue to use tuplespaces for microblogging, but we will adopt a distributed model similar to the one described for message queuing in Section 4.3 of the textbook: each user will have their own separate tuplespace.

Code Provided

The code posted to https://github.com/ProfAvery/cpsc551/tree/master/tuplespace implements a tuplespace and XML-RPC adapter similar to the ones specified in Project 1, along with some additional features.

In particular, this tuplespace sends notifications via IP multicast for the following events:

1. The tuplespace process starts
2. A tuple is added to the tuplespace
3. A tuple is removed from the tuplespace

A client-side stub class named TupleSpaceAdapter is provided so that all code for this project can be written in Python 3. See README.md for additional information.

Create configuration files with unique tuplespace URIs and adapter ports for each user alice, bob, and chuck, and then tuplespace/adapter process pairs for each.

Logging events and recovering tuplespaces

Tuplespaces are not persistent, so their contents will be lost when the tuplespace process ends.

Write a recovery server in Python 3 that joins a multicast group for the tuplespace and logs all write and take event notifications. When a start event notification is received, re-execute the write and take events to restore the state of the tuplespace.

Your server should allow for multiple tuplespaces to be running and sending notifications to the same multicast group. Each tuplespace should be restored when it starts.

A first approach: Naming and Delivery

Write a command-line client in Python 3 that users can use to post messages to a microblog. Messages should be delivered to each user's tuplespace.

Instead of hard-coding XML-RPC URIs into the client or requiring the user to specify the URIs for delivery on the command-line, you will look up the users dynamically.

Write a naming server in Python 3 that will watch for tuplespace and adapter start event notifications and record their URI endpoints. The naming server should persist values using its own separate tuplespace and XML-RPC adapter.

An alternative: Replication

Having the delivery client look up the names and URIs of each tuplespace is cumbersome and error-prone. It would be better if a user could post only to their own tuplespace and have their messages replicated automatically to all other tuplespaces.

Write a tuplespace manager in Python 3 that will join a multicast group and listen for event notifications from other tuplespaces. When a tuple is written or taken from another tuplespace, your server should write or take the same tuple from the tuplespace it is managing.

Since your tuplespace manager receives event notifications, it can replace the recovery server as well as the naming server. Log event notifications, and when the tuplespace is started, re-execute the write and take events to restore its state.

Evaluating the approaches

Include in your submission a short report comparing the two approaches in terms of reliability, fault tolerance, and correctness (where correctness means "each user's tuplespace contains all microblog messages posted". Consider at least the following situations:

- A tuplespace goes down
- An adapter goes down
- A tuplespace is restarted
- A new tuplespace (e.g. dan) joins the group

Include additional factors such as the protocol used for event notification.

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 and any other relevant artifacts to the `project2/` 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.