Project 2 Bayou

Chao Xie, Lorenzo Alvisi

April 18, 2013

1 Bayou Application

In this project, you will implement an eventually consistent store in the style of Bayou that will support a shared playlist. For simplicity, we assume that there is only one playlist with items in the form of {songName, URL}. Each user can perform the following operations on the playlist:

Add Add a new song to the playlist

Delete Delete an existing song from the playlist

Edit Change the URL of a song in the playlist

Bayou will keep the eventual consistency of shared playlist. Note that the system's membership is not static: nodes can freely join or leave the system. Of course, when nodes leave the system, they should follow the "retirement" protocol.

2 Test cases

We list below several test cases that your implementation should handle correctly. This is **not** an exhaustive list of all the tests we will be running—but you can count on the fact we will run at least these!

1 Nodes' logs should be well formed. As nodes create updates and communicate with each other, it should be possible to "freeze" the execution and examine the nodes' logs. Logs should be consistent: every stable log should be a prefix of the longest stable log. Moreover, the tentative logs should be in the correct order (and of course the correct mechanism for transitioning updates from tentative to stable should be in place).

2 The system should guarantee eventual consistency. After having created and exchanged updates for a while, nodes should stop creating updates and just exchange them with one another. Eventually it should be possible to verify that all updates have become stable and all logs are identical.

3 The system should guarantee the "Read your Writes" property to its clients. You should handle the following scenarion: a client, after having performed a set of read and write operation on a particular database node, should be able to read its own writes even if its connection is switched to a different node.

4 The system should handle nodes that "gracefully" leave the system by running the "retirement" protocol. Here is a scenario you should handle. Say that node R that intends to retire. You should make R communicate with another node (A) by gossiping a number of updates that R has just issued. Then R should retires by writing to itself a "retirement" update. R then should talk to another node, say B, to whom it gives all of its updates, including the "retirement" update. Upon processing the retirement update, B should delete R from its timestamp vector. So far, so good. Now, for the kicker: right after deleting R, B should gossip with A and receive any update that A has. Your protocol should handle R's retirement correctly: in particular, B should not end up re-adding R to its timestamp vector as a result of communicating with A.

3 Commands and Scripts

In order for testing, you need to implement the following commands:

- **startClient(i,j)** Start a client with client id i, which is conneted to server j.

- **clientDisconnect(i)** Client i disconnects with the server.

- **clientReconnect(i,j)** Client i reconnect to server j. We use this command after client i has disconnected with server by calling clientDisconnect(i).

- **pause** Pause the progress of the Bayou protocol, later you can continue the execution of the protocol.

- **continue** Continue after pause.

- **printLog** Print logs of all nodes.

- **printLog(i)** Print log of node i.

• **Isolate(i)** Node i is partitioned from other nodes.

• **reconnect(i)** Node i is connected to all other nodes. It is used when we try to recover the connections after we call partition(i).

• **breakConnection(i,j)** Break the connection between Node i and Node j.

• **recoverConnection(i,j)** Recover the connection between Node i and Node j.

• **join(i)** Node i joins the system.

• **leave(i)** Node i leaves (retires from) the system.

These commands are not only convenient for demonstrating the correctness of your system, they also very useful for your debugging. If you have script to demo the open test cases, it will save a lot of time when you meet with me.

4 Implementation

- You can use any language you are familiar with.

- Before you start implementation, make sure you have already understood the protocol.

- One way to demonstrate some cases is to instrument your system so that it can operate in "slow mode"—in which a delay is insterted between consecutive steps.

- If you have any question, contact me by email(xiechao89@gmail.com) or come to my office hour.

- Start early!

- Start early!

- Did I mention it? Start early!