ELEC 477 Distributed Systems Assignment 2 - RPC Discovery and Error Recovery

Due March 4, 2024, 23:99.

1. Purpose

In this assignment you will add a flat namespace to your RPC simulation and make the remote procedure calls more robust.

2. Network Simulator Changes

The simulator has been updated for this assignment. It now has a simulated DNS. When your main program creates an instance of a node with a name, sets the address of that node (and any other parameters) and calls init, the server name and address will be saved in a map. You can then use getaddrinfo() to retrieve the address of another node by name.

To ensure the simulated DNS is cleaned up, call the execute() method from main. It will call the start method and clean up once it returns. It also inserts the try catch that was in main, so you don't have to do that any more. More details are in the 477 Network Simulator V2 Changes document on OnQ.

3. Starting point

You can choose to modify your solution to assignment 1 to use v2 of the network simulator, or start from the posted solution to assignment 1 that uses v2 of the network simulator.

4. Name Service

The first goal of this assignment is to run multiple KV servers with different "names". Thus one KV service might be a KV service for apartment listings, the second might be an KV service for stock listings.

First you must implement the service directory server. This is similar to the kv server, but the key is now a string (i.e. the human readable name of the service), and you have to store two values, the name of the server(string) and a port. The operations are to register a service (i.e. put), lookup or search for a service (get) and delete a service.You will have to create the appropriate client interface (a client sub and 3 methods: register, search and delete), and define any necessary data structures (similar to struct kvGetResult). You will have to write a service for the server. You do not have to use a persistent store such as gdbm (ndbm) as this would require you to serialize the server name and port into a string to store in database file. You can use a C++ map to associate a two element structure with a string.

The client stub class for the service directory will use a static name for the server similar to the current assignment 1 solution for the kv servers. You should write a simple test client to test that your directory service works.

5. Use the Name Service in the kvRPC

Add another member variable (e.g. ***svcName***) in the service subclass for the human name of the service (e.g. "apartments"). The existing ***name*** field in the base class (***Service***) is used by the network library for debugging messages. Add a method in your server and service classes to set the name of the service. Also add a member variable for the port number and a method in the server and service class so that the main program can set the port number your kv service will listen on. This will allow you to test that your kv client stub can properly find the service regardless of port.

When your service starts it will in turn call the RPC for the directory service to register the service name, the name of the server (use the nodeName() method) and the port it is listening on. At the end of the service (after the alive loop), it will delete the service from the directory service.

Your client stub for your kv service will also have a field for the service name that can be set by the client node subclass. The client stub (not the client node) will use that name in its initialization to call the directory service server to find out the address of the server for the named kv service, and then open the socket connection to that named address.

The actual kv put/get routines in the service should remain unchanged. The only change to the client stub is to use an rpm to the directory service to find the address of the server.

As an example, you should be able to create a kvserver with the name ***server1*** and a human service name of "busRoutes", and a kvserver with the name server2 and a human service name of "stocks". Assuming server1 has been told to use port 4250 and server2 has been told to use 5390, the directory service should have the entries (busRoutes,(server1,4250)) and (stocks,(server2,5390). So the client node that wants to add and remove entry from busRoute will do an searchService("busRoutes") and get back the result and connect to port 4250 on node server1.

6. Improve the RPC semantics

The current put method implements the at most once semantics. That is the put method may or may not succeed. The last part of this assignment is to change the semantics to at least once. This means that the client stub will have to retry the message until it gets an affirmative result from the associated kv server. Since the put operation is idempotent, a second execution of the RPC does not change the operation of the program.

You should have a constant somewhere that is the maximum number of tries that the put() method client stub will try. If the maximum number of tries is exceeded or there are unrecoverable network errors (e.g. cannot open a socket), then return false.

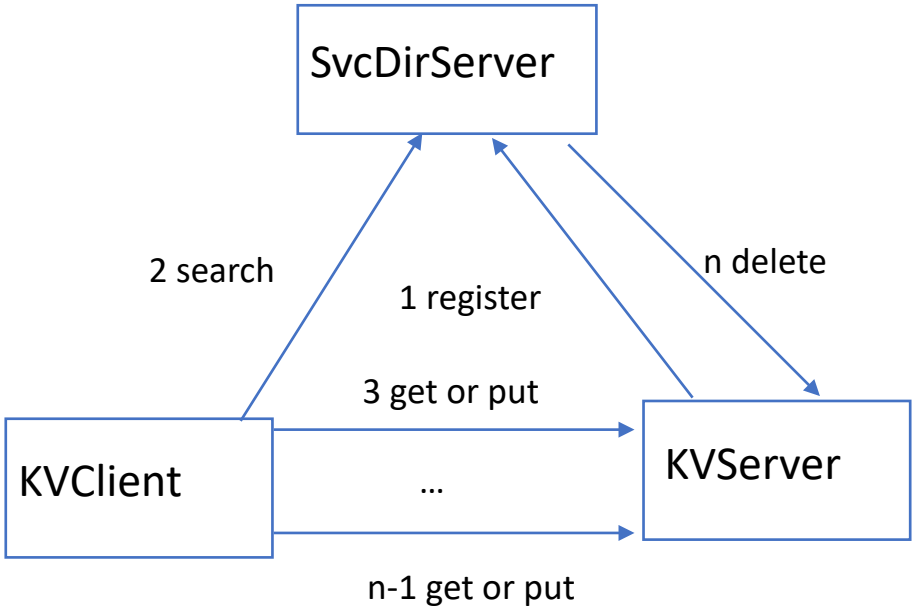
Your search and delete methods for the name service should also implement the at least once semantics.

7. Testing

You have already tested the implementation of the key functionality in assignment 1. I mentioned earlier that you should test the service server separate with a client that simply stores and retrieves service information. For testing the integrated system, you should test two different kv servers running as different service names, and that the clients correctly find them by service name. You should check that when a kv server exits, that the name of the service is no longer in the service server.

8. What to hand in

As with assignment 1, you should have multiple test cases and both a document explaining how you solved the program and a test document that explains how you tested your system to ensure it works.


SvcDirServer
2 search
1 register
n delete
3 get or put
KVClient
...
KVServer
n-1 get or put