Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

Testing Document – ELEC 477 – Lab 3

The testing strategy were designed to cover the main functionalities introduced in this lab, specifically focusing on the primary backup service, security methods, and dispatch call methods of the KV Service with the consistency model now implemented. The test cases below detail the methods used to validate our implementation.

Test Case: Trace Methods

The goal for this test case was to verify the creation of the backup servers/replicas and ensure the system still maintained the same functionality as assignment 2, with the added security measures the consistency model brings. This encompasses instantiating a primary and backup server along with the client and establishing the connection between all of them using a UDP network. The main modification made to ensure a clear difference between a replica and Primary Server upon creation by making use of the primary status Boolean and not including the backup server within the directory, while still ensuring that they had different GDBM files.

The steps for this test are as follows (refer to Figure 1 and for main.cpp implementations of these steps):

1. Start the service directory server.
2. Start the KV server with a service nickname and port. Additionally, set the Boolean to ensure it's the primary copy to 'True' and name of its to be created replica to backup1.
    a. This will ensure the Primary Server is correctly registered in the directory and the information of its replica is correctly set.
3. Start the KV backup server and set its Primary Server to the server previously created.
4. Start a KV Client and provide it with the nickname of the KV Primary Server.
5. Let the Client execute.
    a. This involved putting and then getting key-value records from the set of KV servers. Client will communicate with the Primary Server only, while the Primary Server ensures its backup server is also up to date.
6. After results, stop all services/servers.

If functioning correctly, the expected outcome would be the Primary Server successfully replicates information to its backup server, and the put request is successfully completed once the consistency is ensured.

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

```cpp
int main(int argc, char * argv[]){
    // handle command line arguments...
    int res = network_init(argc, argv);
    std::stringstream ss;

    // start all of the servers first. This will let them get up
    // and running before the client attempts to communicste
    std::cout << "Main: **********************************" << std::endl;
    std::cout << "Main: starting servers" << std::endl;
    std::cout << "Main: starting directory server" << std::endl;
    shared_ptr<SDServer> sdServer = make_shared<SDServer>("Directory1");
    sdServer->setAddress("10.0.0.1");
    sdServer->init();
    sdServer->startServices();

    //start the primary server
    std::cout << "Main: starting primary kvserver" << std::endl;
    shared_ptr<KVServer> kvServerPrimary = make_shared<KVServer>("kvserverPrimary");
    kvServerPrimary->setAddress("10.0.0.3");
    kvServerPrimary->setDBMFileName("kvserverPrimary");
    kvServerPrimary->setNickname("snow");
    //kvServerPrimary->setSvcName("kv1");
    kvServerPrimary->setPort(5193);
    kvServerPrimary->setIsPrimary(true); // Mark this server as primary
    kvServerPrimary->addBackupName("backup1");
    kvServerPrimary->addBackupPort(5194);
    kvServerPrimary->init();

    //start the backup server
    // Initialize a Backup KVServer (you can add more backups following a similar pattern)
    shared_ptr<KVServer> kvServerBackup = make_shared<KVServer>("backup1");
    kvServerBackup->setAddress("10.0.0.4");
    kvServerBackup->setDBMFileName("backup1");
    kvServerBackup->setPort(5194);
    kvServerBackup->setIsPrimary(false); // Mark this server as backup
    kvServerBackup->setPrimaryServer("kvserverPrimary");
    kvServerBackup->init();
    kvServerPrimary->startServices();
    kvServerBackup->startServices();
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
```

Figure 1: Test Case 1 main.cpp code (part 1)

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

```
std::cout << "Main: **********************************" << std::endl;
std::cout << "Main: init client" << std::endl;
std::this_thread::sleep_for(std::chrono::milliseconds(1000));
shared_ptr<KVClient1> kvClient = make_shared<KVClient1>("kvclient");
kvClient->setAddress("10.0.0.14");
kvClient->setServerName("snow");
kvClient -> init();
std::this_thread::sleep_for(std::chrono::milliseconds(1000));

std::cout << "Main: **********************************" << std::endl;
std::cout << "Main: starting client" << std::endl;
vector<shared_ptr<thread>> clientThreads;
{
    // need a scope for the lock guard.
    // if this doesn't work put it in a function
std::lock_guard<std::mutex> guard(nodes_mutex);

shared_ptr<thread> t = make_shared<thread>([kvClient](){
    kvClient -> execute();
});

clientThreads.push_back(t);
nodes.insert(make_pair(t->get_id(), kvClient));
names.insert(make_pair(t->get_id(),"kvclient"));

}

// wait for clients to finish
std::cout << "Main: **********************************" << std::endl;
std::cout << "Main: waiting for clients to finish" << std::endl;
vector<shared_ptr<thread>>::iterator thit;
for (thit = clientThreads.begin(); thit != clientThreads.end(); thit++){
    shared_ptr<thread> tmp = *thit;
    tmp->join();
}

std::this_thread::sleep_for(std::chrono::milliseconds(1000));
std::cout << "Main: **********************************" << std::endl;
std::cout << "Main: calling stop services on servers" << std::endl;
kvServerPrimary -> stopServices();
```

Figure 2: Test Case 1 main.cpp code (part 2)Test Case 1 Output

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

```
paufk@DESKTOP-EMADDII:/mnt/c/Users/paufk/networksim/simulatorExamples/team25/a3$ bin/assign3
Main: ************************************
Main: starting servers
Main: starting directory server
Starting service: Directory1
Main: starting primary kvserver
Starting service: kvserverPrimary.KV_RPC
Starting service: backup1.KV_RPC
KV Primary Server: ******************************************
KV Primary Server: kvserverPrimary registering self as snow
SDStub: attempting to register new service
SDStub: successful send to socket: SDService::start() successful recieved in while
SDService: registering new server in directory
registration -- COMPLETE
1000
KVServiceServer: **BACKUP Request**- Backup Name: backup1 @Port: 5194 for primary server: kvserverPrimary.KV_RPC
KVServerStub: Creating KVServerStub for backup: backup1 on port 5194
KVServerStub: **BACKUP copy complete** - exiting KVServerStub Constructor
Main: ************************************
Main: init client
Main: ************************************
Main: starting client
Main: ************************************
Main: waiting for clients to finish
SDStub::searchForService
attempting to search for service
service nickname: snow
successful send to socket: 1000
SDService::start() successful recieved in while
SDService: searching for server in directory
in SDService::searchForService looking for: snow
service for server (snow) found in directory
search results good, found server - kvserverPrimary and port - 5193
search message received: good
put message requested
put result is 1
[Primary Server] Forwarding PUT request to 1 servers.
[Primary Server] Forwarding to Backup Server (Name: backup1, Port: 5194).
[Replica Server] Received PUT request.
[Replica Server] Key: 25, Value Length: 18, Value Snippet: 'This is  '...
KVServerStub: Socket setup successful
KVServerStub: kvPut called with key: 25, vlen: 18
put message requested
put result is 1
[Replica Server] PUT request processing result: Success
[Primary Server] Forwarding result for Backup Server: Success
[Primary Server] Completed forwarding PUT requests to all backups.
status is 1
get message requested
leaving get stub
status is 1
00 54686973 20697320 00206120 74657374 This is . a test
10 2121                                 !!
12
Main: ************************************
Main: calling stop services on servers
Main: ************************************
Main: waiting for threads to complete
Main: ************************************
Main: stopping directory
```

Figure 3: Test Case 1 terminal output.

As seen above, the test case was successfully executed. Both the primary and the backup server were successfully created. Only the Primary Server is registered within the Directory service. Then, after requesting a put, the request is forwarded from the Primary Server to the Replica Server with the correct message as seen by the message snippet. The Replica Server then sends the acknowledgement message to the Primary Server so the Primary Server can confirm the replication is complete. Only then confirming the put message request has been successful and confirming this with the client. All clients and KV servers stopping after this activity is complete.