Group 6: Team Slackers

Design Document for Fault Tolerance Evaluation

1. **Chief experimenter: Hyunwoo Kim**
2. **List of client invocations that we will measure:**

Method	*One Way?*	*Database Access?*	*Request Size*	*Reply Size**
enterLot	N	Y	8	4 * array length
exitLot	N	Y	4	0
getClientID	N	N	0	4
getOtherLotAvailability	N	Y	4	4 * array length
getLots	N	Y	0	4 * array length
moveUpLevel	N	Y	4	4
moveDownLevel	N	Y	4	4
getCurrentLevel	N	N	0	4
getMaxLevel	N	Y	0	4
getMinLevel	N	Y	0	4

* Size of the reply before the experiment padding is added

3. **List of implementation changes required for fault-tolerance evaluation:**
    1. **Two-way invocation**
        1. For each invocation, all methods in the server that we're going to measure return a reply that the client receives
    2. **Tunable reply size**
        1. Add new structures in the IDL and change the method return types in ClientManager to permit the passing of padded return values from the server to achieve a test's reply size parameter
        2. Modify calls to the ClientManager methods to use the real return values embedded within the padded return values
    3. **Inter-request time**
        1. The methods in the FaultTolerantClientManager on the client side will wait for the inter-request time's value after notifying the Logger that the method call completed.
    4. **Other changes**
        1. Add command-line parsing in the server and client main() methods to permit passing of the test parameters – number of clients, inter-request time, and reply size – from the test scripts
        2. Add a text user interface on the server to permit both manual log flushing and server shutdowns

3. Pass client host names from the client to the server in the getClientManager() and getExistingClientManager() methods so that the server knows the host name of its caller for the client host name probe
4. All server-side method implementations in ClientManagerImpl, except for closeClientManager(), will inform the probe logger of when each method call begins and when each method call ends along with the called method name and the calling client's host name. Normal returns and exception throwing will inform the probe logger that the method call ended. Note that getClientManager(), getExistingClientManager(), and poke() exist in the ClientManagerFactoryImpl and are not measured
5. Create a Logger class that the server and client call to notify of method entries and exits. Internally, this class measures specific time points in microseconds. This class stores its logging data in a preallocated array until flushed either manually from the server's text user interface or automatically from being filled. Flushing writes all the logging data to specially-named log files on the disk.
6. Create a LogEntry object that the Logger class uses to represent a single log entry. When measured methods notify the Logger that the method is complete, those methods pass in the LogEntry object instance returned when that method notified the Logger that a method call was starting
7. Flush the Logger's logs to log files on the disk during the client's shutdown
8. The ClientManager methods implemented in the FaultTolerantClientManager class in the client, except for closeClientManager(), will inform the Logger of method call starts and method call completions.
9. Ensure that the existing client and server startup scripts run Java in the foreground and pass all command-line parameter arguments to Java with $@
10. Create scripts that can start up the desired number of servers and clients, run them, and collect the logged results

4. List of scripts required for fault-tolerance evaluation and the design for fault-tolerance evaluation (e.g., how you're planning to put the scripts together, gotchas, etc.):

Script Location	*Type*	*Description*
server/server	bash	Starts a single server instance in the foreground (already implemented)
client/client	bash	Starts a single client instance in the foreground (already implemented)
tests/client-test	text	Command list to pipe into the client's stdin stream to cause the client to test the remote methods that we want to test
tests/runall	bash	Runs all 48 test configurations by calling tests/runone with different command-line parameter arguments

Script Location	*Type*	*Description*
tests/runone	bash	Runs a single test configuration as configured by the command-line parameter arguments. This is accomplished with the following actions: 1. Start a server instance on each server listed in tests/servers using a backgrounded SSH session, redirecting stdin to a file descriptor that the script later writes into to tell the server to flush logs and exit 2. Start a clients instance on each computer listed in tests/clients in the background using SSH and, on the remote side of each SSH connection, pipe tests/client-test into the client as its stdin 3. Wait for the clients to exit 4. Inform the servers that they need to exit by, one at a time, foregrounding and sending the exit command
tests/servers	text	Lists the name of each server to start a server on in separate lines
tests/clients	text	Lists the name of each computer to start a client on in separate lines

5. **Plan at this point:**
   1. Finish the code modifications (they are almost complete)
   2. Implement the scripts
   3. Run the tests/runall script to produce the output log files in client/ and server/ for all 48 configurations
   4. Analyze the results