Latency Analysis

Team 6: Slackers
Steven Lawrance
Puneet Aggarwal
Karim Jamal
Hyunwoo Kim
Tanmay Sinha

Time Evaluation for Fault Tolerant Real-Time Park'n Park

To determine which parts of Park'n Park should be optimized to provide bounded fail-over in a high-performance environment, we added extra probe files to measure the operations that take place during fault recovery on the client side.
We ran the system with these probes in place and the following configuration:

- 20 faults injected
- 10,000 invocations of the getLots() method
- Reply size of 256 bytes
- One client
- Two servers in a passive replication arrangement (one primary and one backup)
- 20 millisecond inter-request time
- 17 seconds between server-kill fault injections

The following chart shows the time that each invocation took:

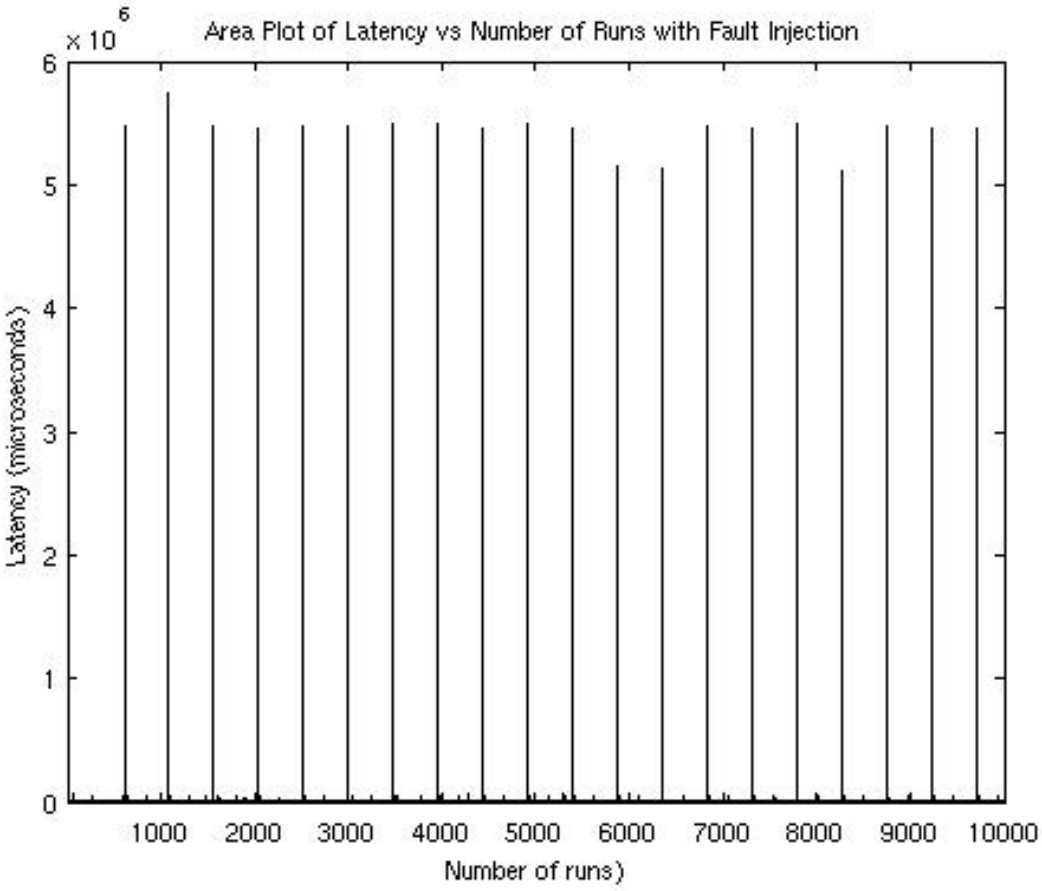

Figure 1: Latency vs. Number of Runs with Fault Injection

After analyzing the results, we determined that the biggest two contributors to the fault recovery process are the client's delay while waiting for the name service to register a new primary server and the time that the client has to wait while the client's manager instance is recreated on the server.

We looked at the spikes in figure 1 and placed the components of those spikes into a pie chart, shown in figure 2.

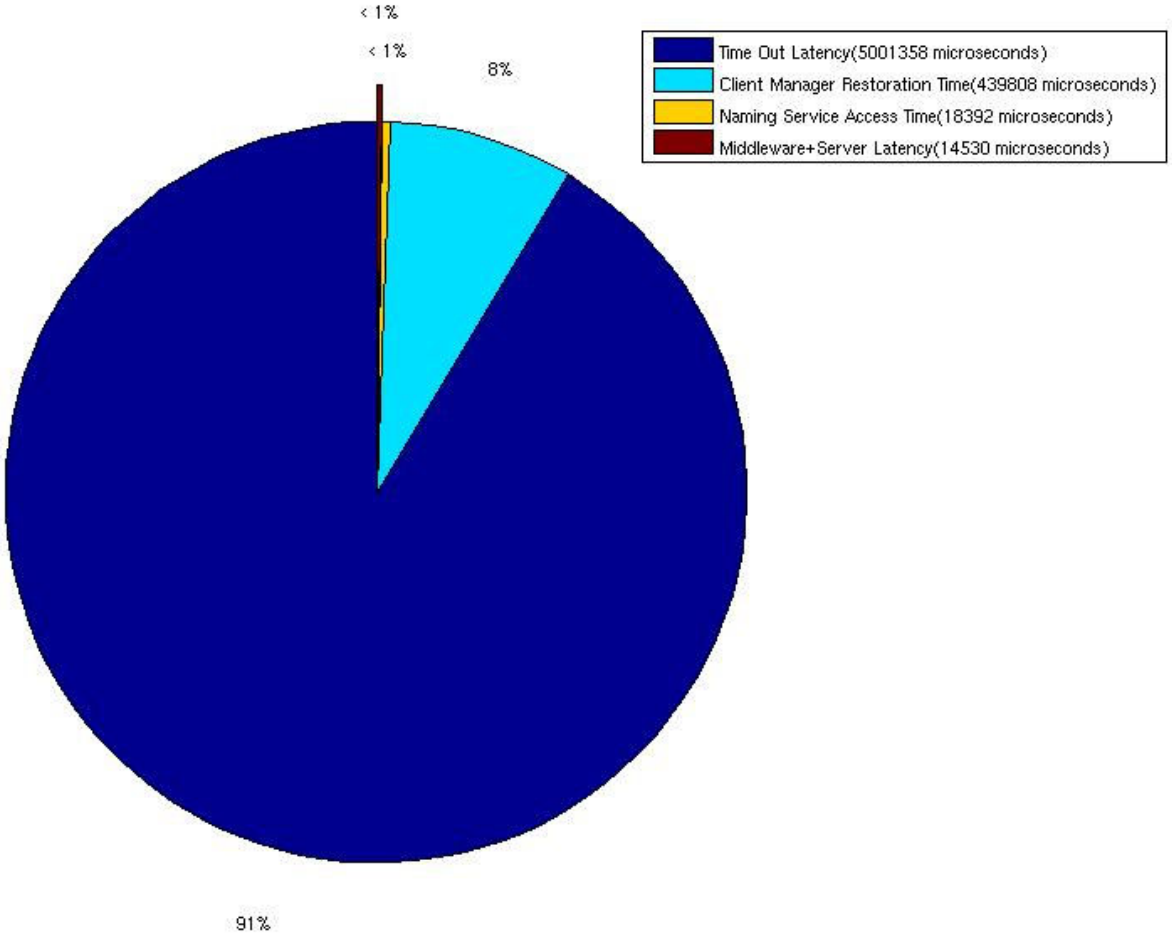

Figure 2: Chart of Round-Trip Latency during Fault Injection

When the client detects a fault, it immediately attempts to retrieve the new primary server from the name service. If the primary server that it retrieves does not work and this happens twice, then the client will wait for five seconds before it tries to retrieve the new primary server again. During that time, the replication manager notices that the server is down, chooses a new primary server, and then registers that new primary server with the name service. When the client tries again after its five second wait, it is able to retrieve and successfully use the newly-chosen primary server.

Unfortunately, this five second delay apparently is too long as seen in the pie chart. In our tests, it is able to successfully get the new primary server after one delay cycle, though if the new primary fails before the client finishes its wait, then the client could wait for 10 or more seconds.

The client manager restoration time is the time that the client has to wait for the server to restore the client's session object in the server. This includes the establishment of a new JDBC connection to the database server for that client (each client manager instance on the server has its own JDBC connection), the creation of some prepared statements, and some queries to ensure that the client is valid.

The time that the client spends waiting for the name service operations is logged as the naming service access time.

Because the server latency and the middleware latency are very small, we combined them into one value for the pie chart. This captures the time that the client waits for the server to run the getLots() method and return a value to the server as well as any middleware overhead in that process. This does not include the client manager restoration time that we captured separately, which primes the CORBA and database connections for speedy access to getLots().

In addition to the times that we charted, we also calculated that the client spends about 758 microseconds performing other operations in the fault recovery path such as method calls, loop iterations, variable initializations, and if-statement branch condition checks. To reduce the time that the client spends in its fault recovery process, we propose the following strategies, ranked in decreasing order of importance relevant to our findings:

1. Reduce the time that the client spends waiting for the replication manager to register a new primary server.
2. When a fault is detected, have the client automatically choose another server using IOGRs (or something similar) and cloning rather than wait for the replication manager to register a new primary server.
3. Pre-create TCP/IP connections to all servers so that the TCP connection overhead and client manager restoration processes are less likely to be in the critical path of fault recovery. The client can also keep these open and, in a background thread, periodically check them to ensure that they are still running. That background thread can also refresh the registered server list from the name service so that the client does not always have to do that during fault recovery.
4. Have the server pre-create its connections to the database and maintain a pool of database connections. When new client manager instances are created, the server and the client won't have to wait for the database connection to come up while creating and restoring client manager instances on the server.
5. Implement client load balancing between the servers.
6. Assume that the client ID given by the client on client manager instance restorations is valid so that the server does not have to check its validity.