Analysis Document for High Performance Real Time-Fault Tolerance Evaluation

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

1. List of client invocations that we measured

Method	One Way?	Database Access?	Request Size (in bytes)	Reply Size* (in bytes)
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

The getLots() method was the only method tested for this phase's experiments.

The average size of the original replies, before serialization, was 16 bytes due to four lots in the system.

2. Analysis of experimental results from our high performance RT-FT evaluation

Strategy 1

Our first strategy was to adjust the client-side fault recovery wait time, which the code calls the fault recovery "timeout," though it's not really a timeout on method invocations. The client has this wait time because the replication manager takes time to inform the naming service of the new primary, and continuous name service polling from the clients makes the system crawl.

We graphed our failover times after running the experiments with different client-side fault recovery wait times.

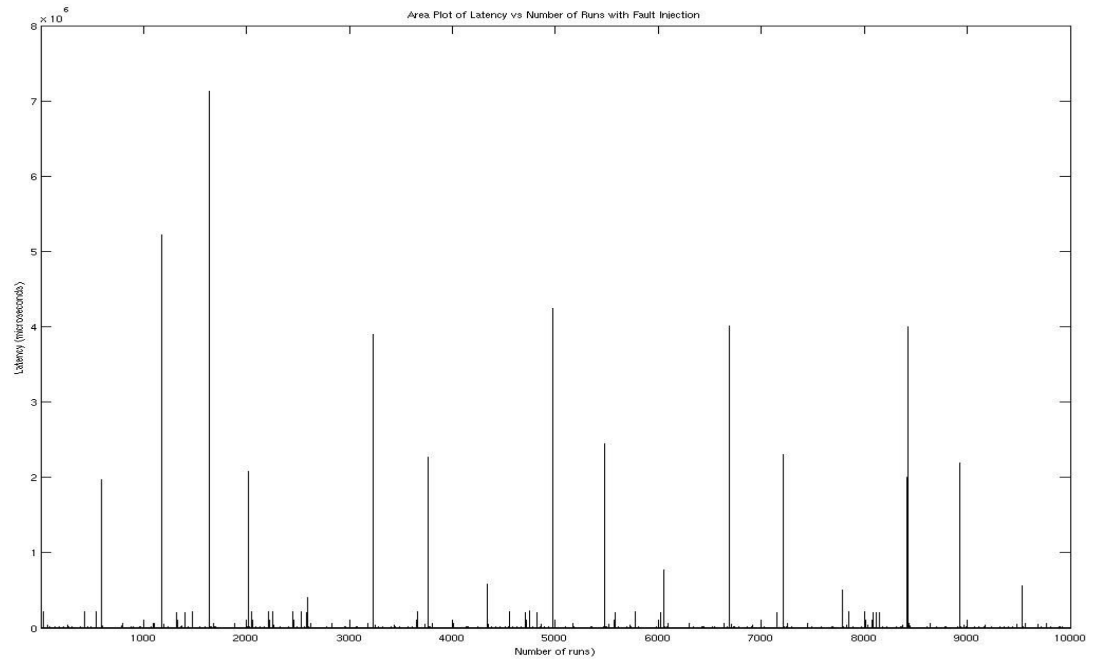

Figure 1. Plot for wait time 0 ms

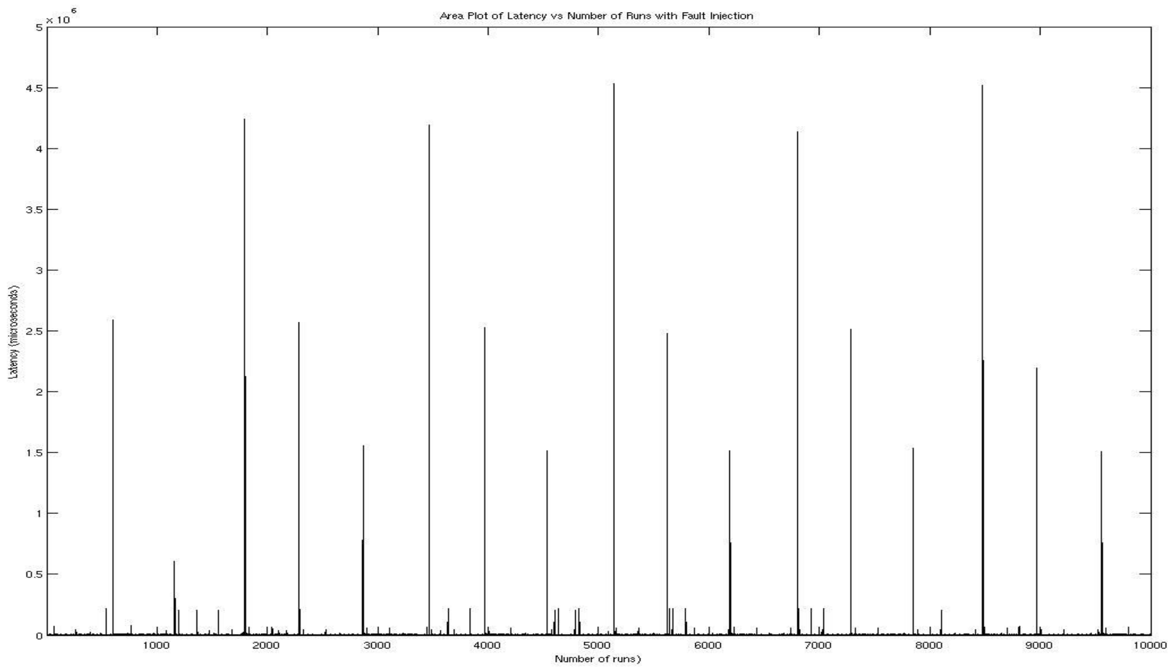

Figure 2. Plot for wait time 1000 ms

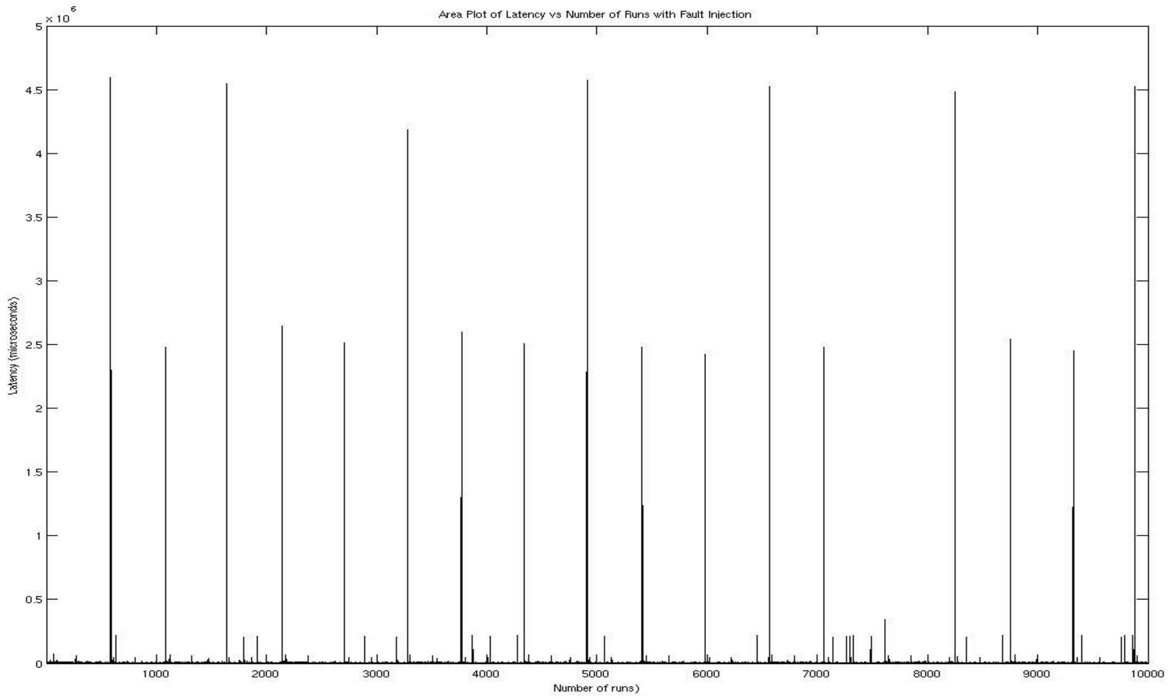

Figure 3. Plot for wait time 2000 ms

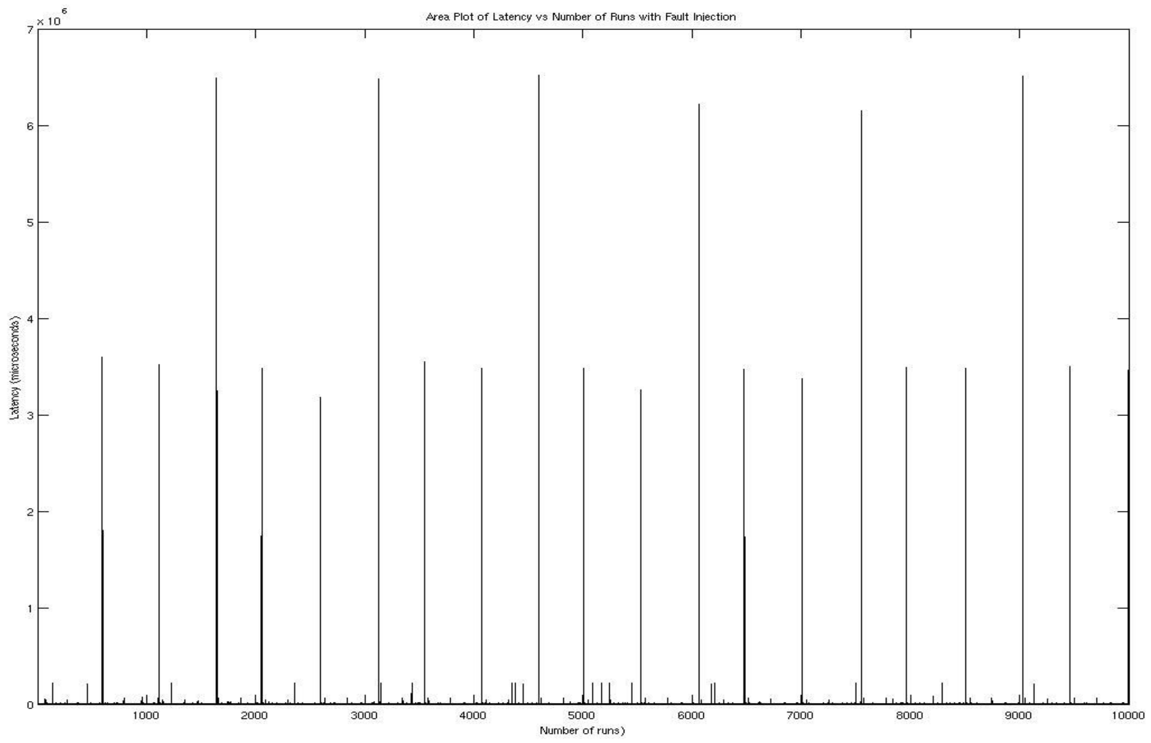

Figure 4. Plot for wait time 3000 ms

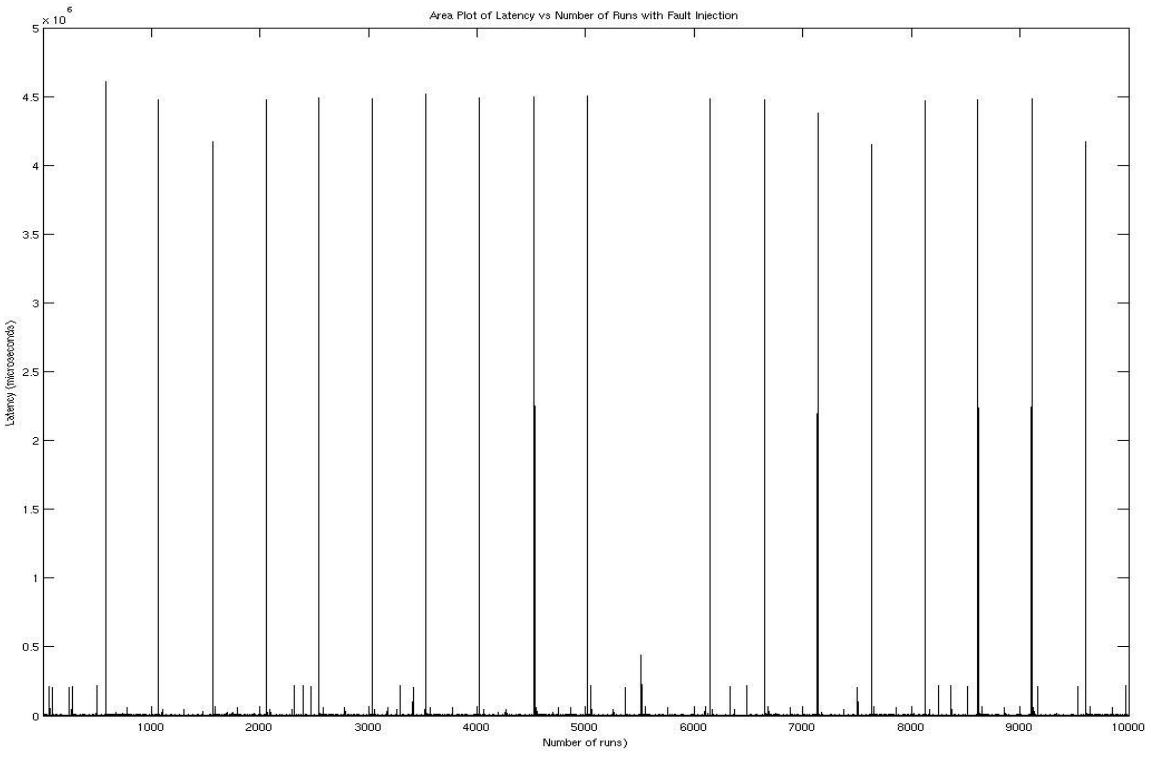

Figure 5. Plot for wait time 4000 ms

After analyzing the available data the observations are as follows

- The best results can be seen with 4000ms wait time.
- Event though there is a lot of reduction in fail-over time for lower values, we can observe a significant amount of jitter.
- The reason for the jitter is that the client doesn't get the updated primary server from the naming service in a timely manner. The replication manager's fault detection wait time, similarly called the fault detection "timeout" in the implementation, was held constant at 5000ms across this strategy's experiments. This means that the replication manager waits for 5000ms after serially checking every registered server.
- Average recovery time is reduced by some amount through this client-side wait time variation strategy (from about 5-6 secs to 4.5-5 sec for 4000ms wait time).
- Our main goal is to have a bounded real-time failover, and simply adjusting the client-side wait time did not adequately provide that.

Strategy 2
Implementing the IOGR (Interoperable Object Group Reference)

In this strategy, the client gets the list of all active servers from the naming service. The client caches this list for fast access, but it has to refresh it if all servers in that cached list have become faulty.

The following graphs show the fault recovery times after this strategy was implemented.

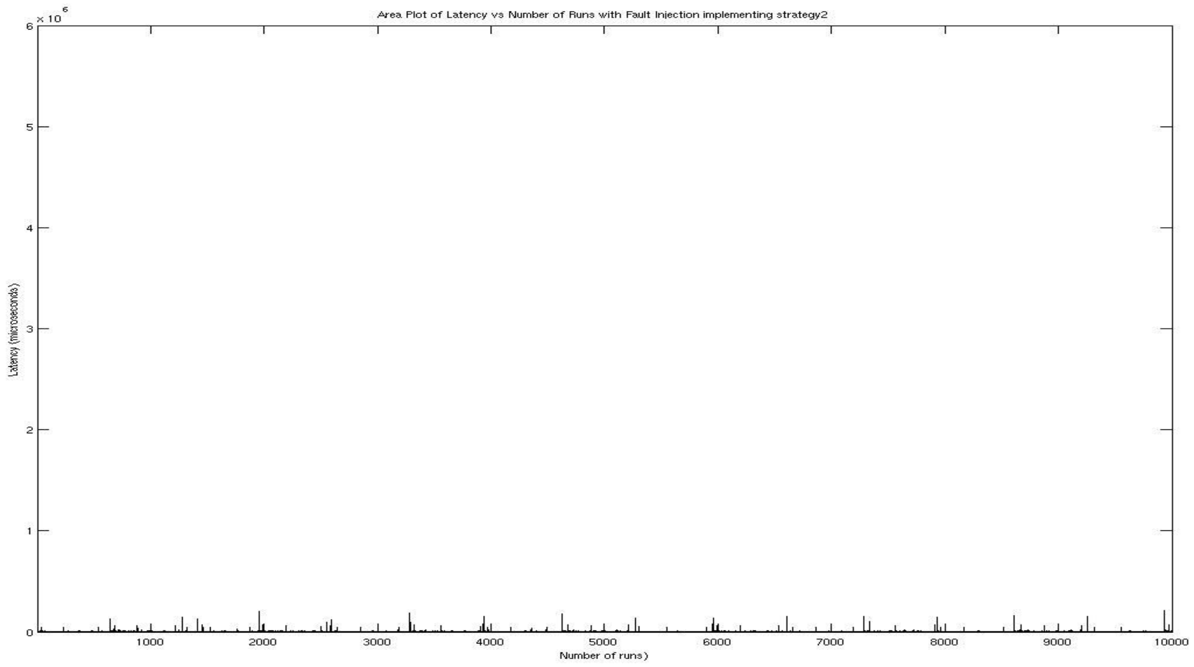

Figure 6. Plot after IOGR strategy (same axis as without any strategy)

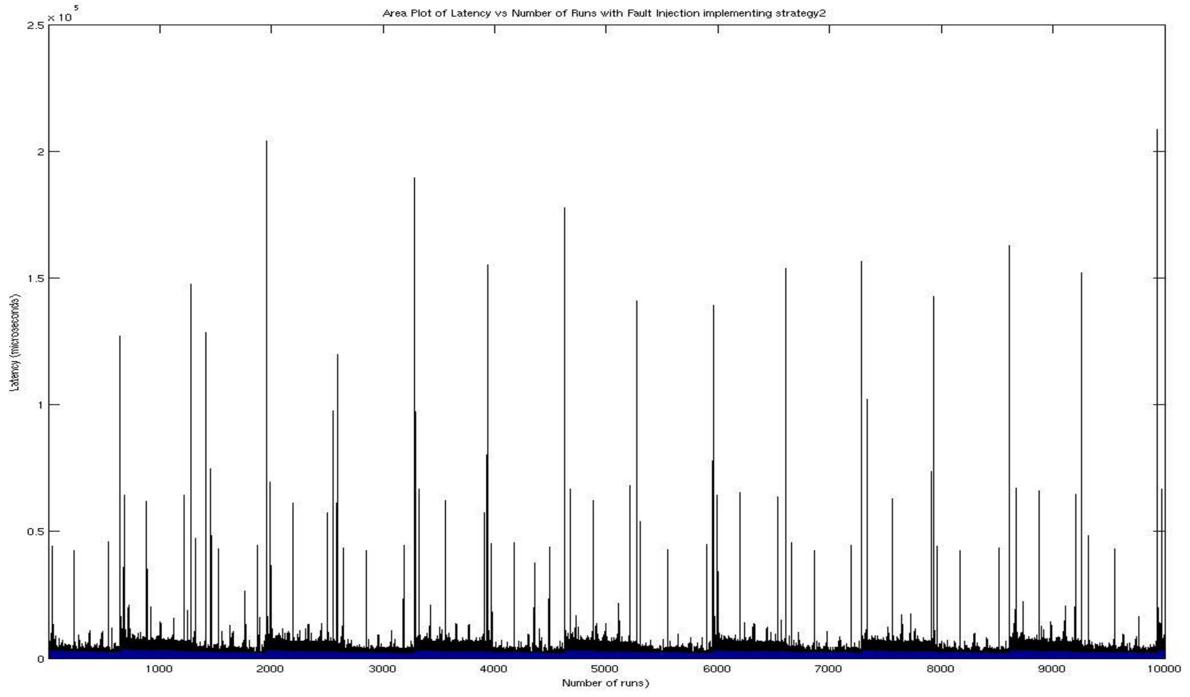

Figure 7. Plot after IOGR strategy (original axis)

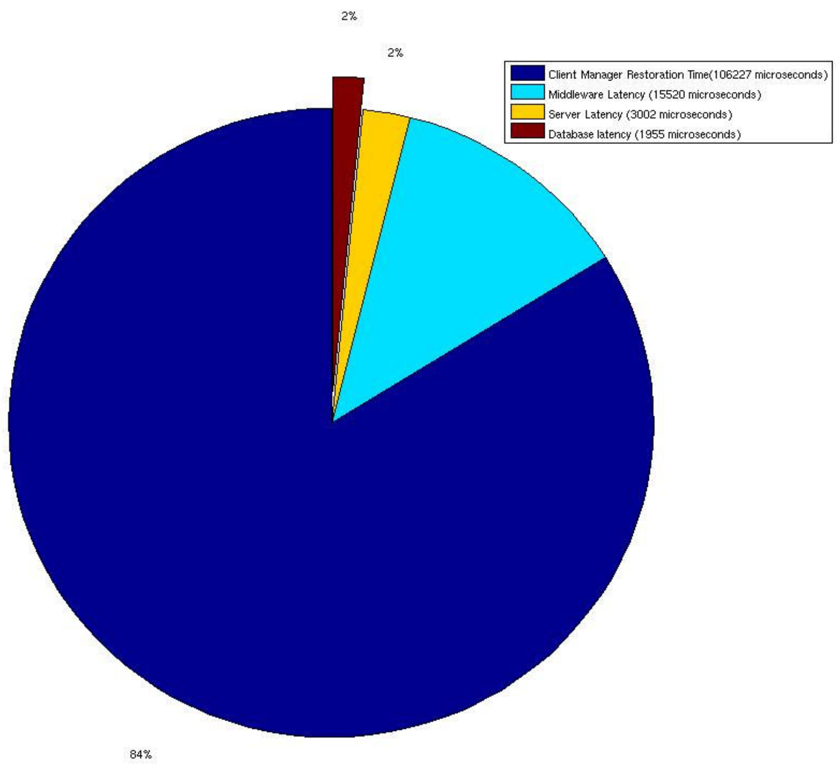

Figure 7. Pie Chart after IOGR strategy

Observations after implementing the IOGR strategy:

- The recovery time is significantly reduced from between 5-6 seconds to less than half a second.
- The time to get the new primary from the naming service is eliminated in the particular case that the pie chart measured, which was a cache hit on the cached server list.
- The most time taken is in restoring the client manager.
- The graph plotted on a different axis shows some amount of jitter. After all the active servers are dead, the client has to go to naming service to refresh the list.

Strategy 3
Open TCP/IP connection

After getting the IOGR implemented, the main component of the end-to-end latency is time taken in obtaining the client manager object. In this strategy, the client maintains open TCP/IP connections with all the active servers. As a result, the time to create a connection is saved, reducing the client manager restore time.

The following graphs show the fault recovery latencies in strategy 3 for 1 client.

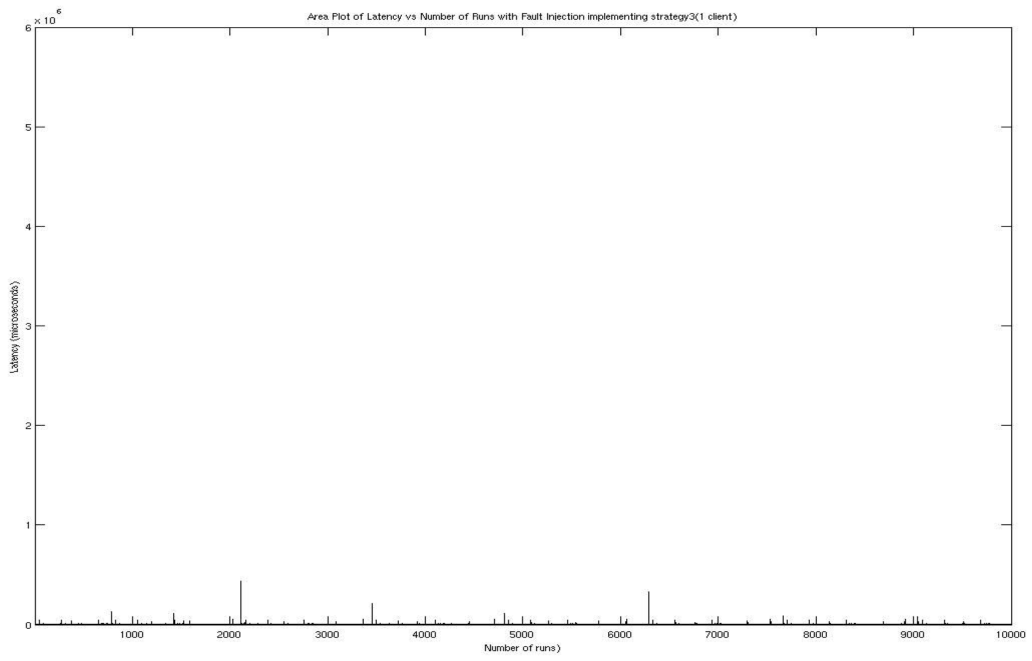

Figure 8. Plot after implementing strategy 3(same axis as without strategy)

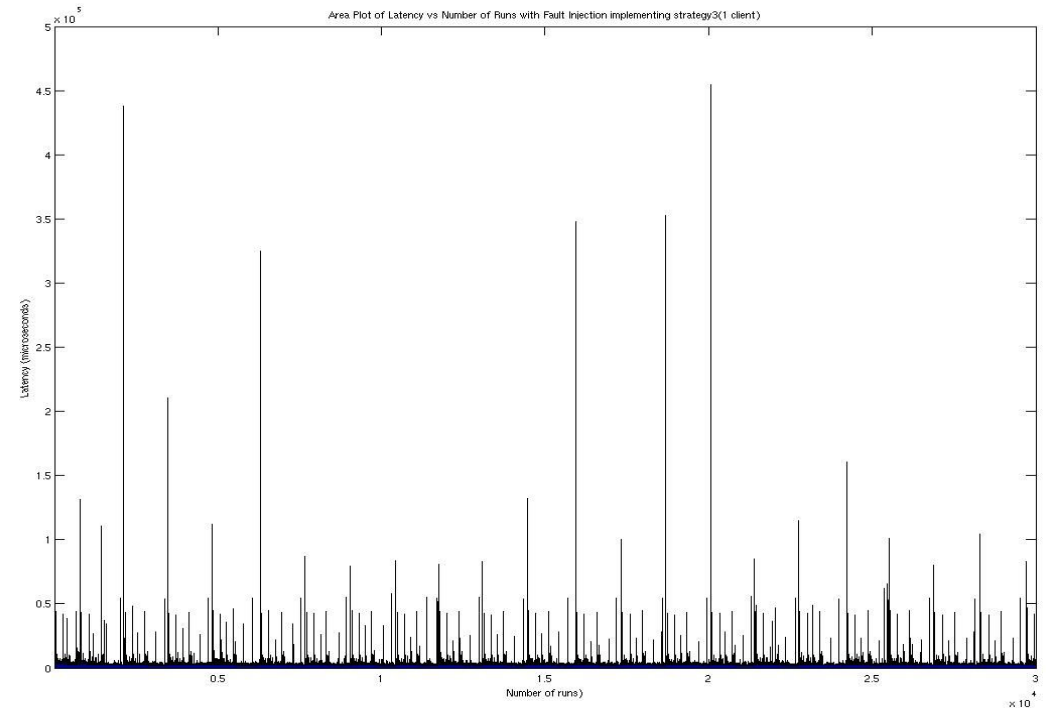

Figure 9. Plot after implementing strategy 3(original axis)

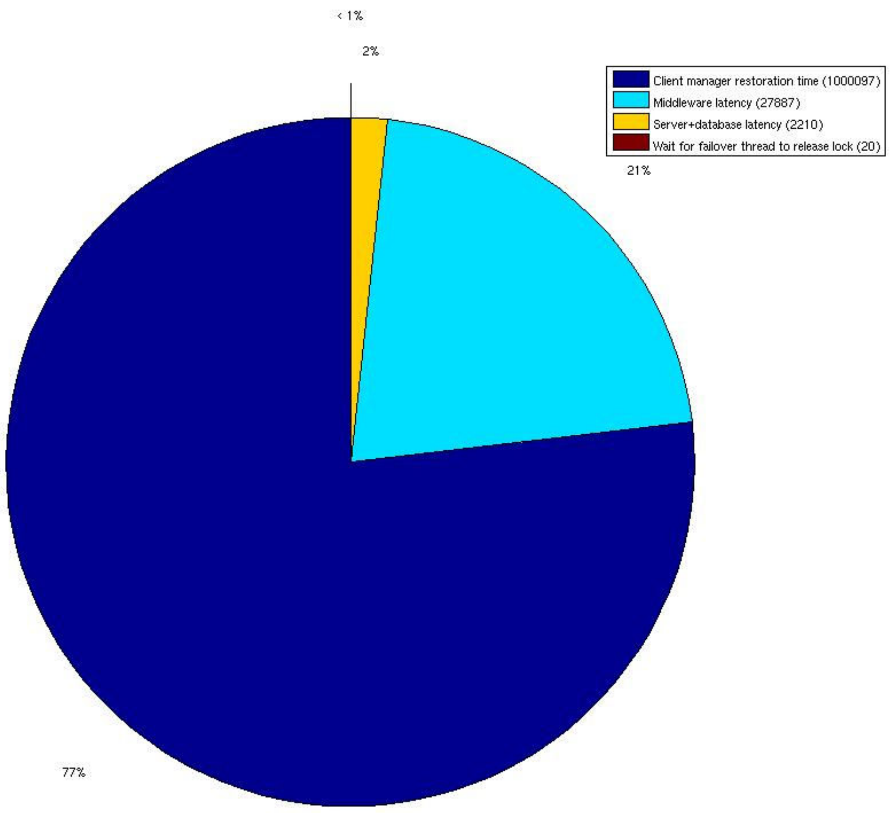

Figure 10. Pie chart after implementing strategy 3

Observations after implementing the strategy 3 as open TCP/IP connections for 1 client

- The recovery time is reduced compared to the IOGR strategy.
- Most of the time taken is still in restoring the client manager.
- There is noticeable jitter when observed on a different axis.
- We had to run the client for 30,000 method invocations to get at least 20 faults injected. We tried to increase the fault injection rate to achieve at least 20 faults in 10,000 method invocations, but the results were not well-bounded. Keeping the client fault injection rate consistent between the 1-client and 10-client experiments helps make the results more comparable.

The following graphs show the fault recovery latencies on strategy 3 for 10 clients.

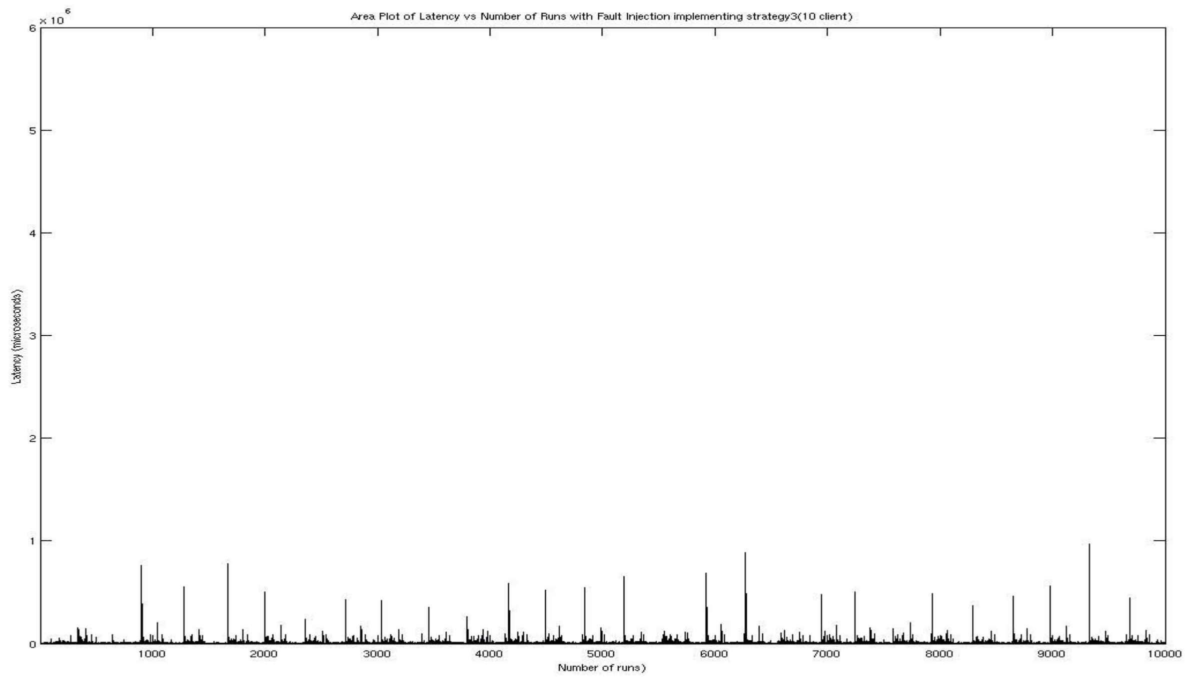

Figure 11. Plot after implementing strategy 3 for 10 clients (same axis as without any strategy)

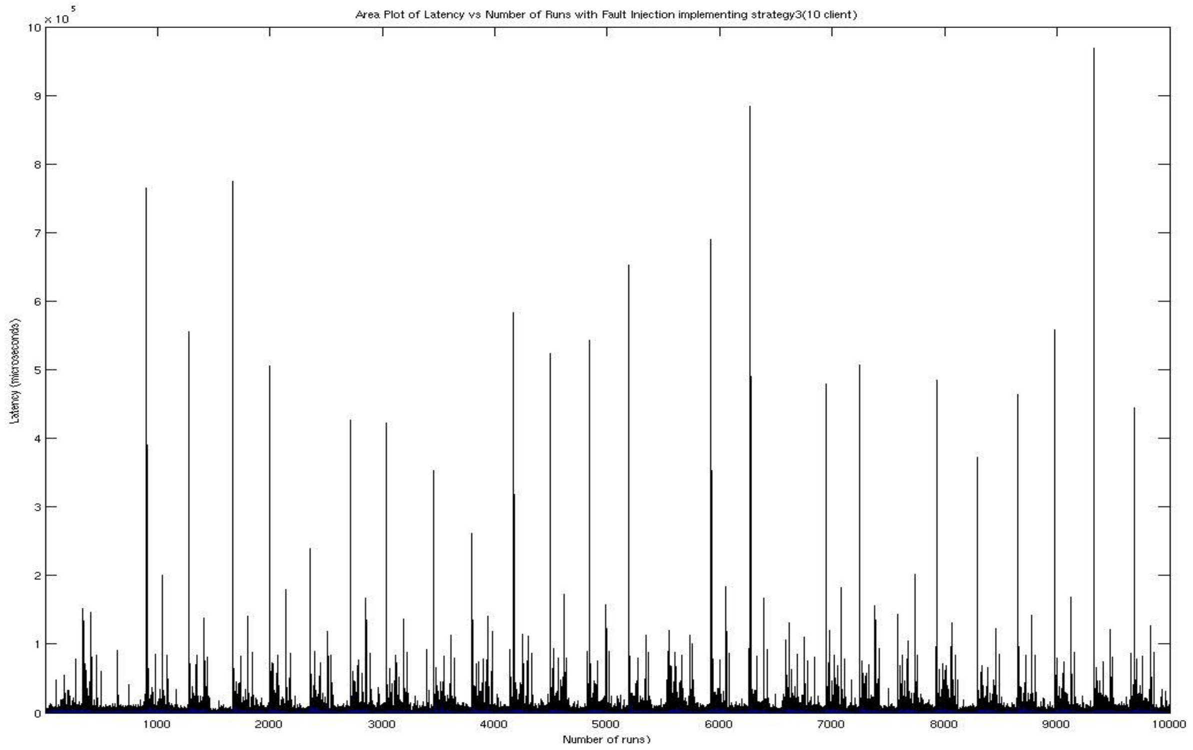

Figure 12. Plot after implementing strategy 3 for 10 clients(original axis)

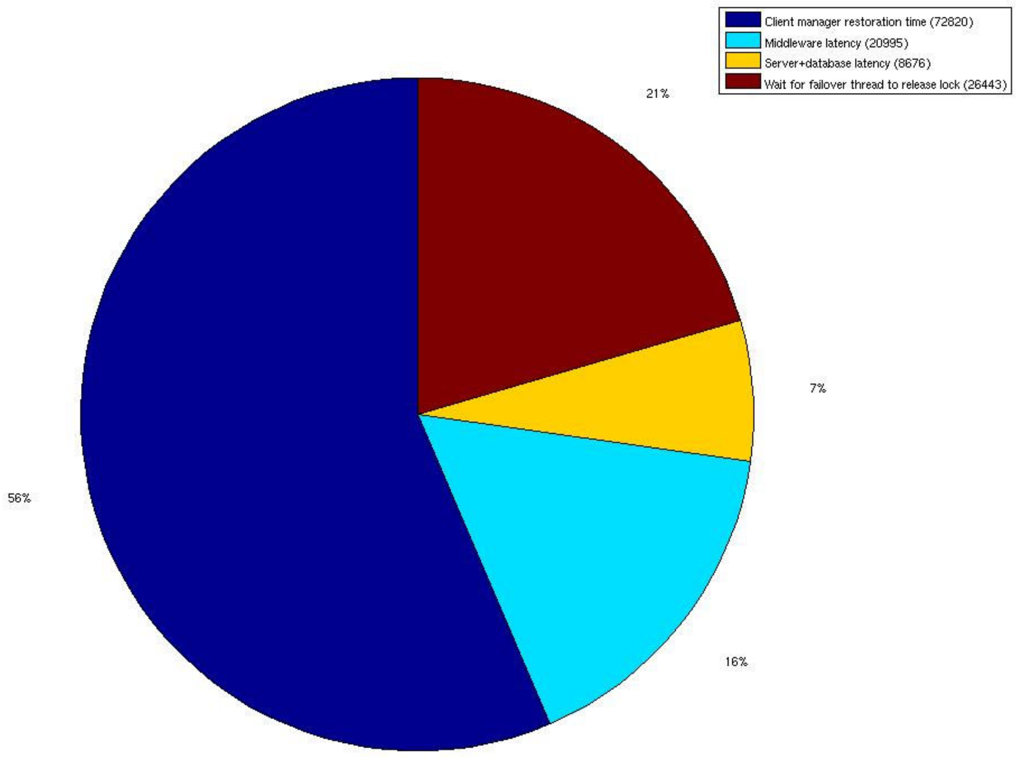

Figure 13. Pie chart after implementing strategy 3 for 10 clients

Observations after implementing the strategy 3 as open TCP/IP connections for 10 clients:

- Significant reduction is observed in the fail-over time.
- Most of the time is still taken in restoring the client manager.
- It can also be observed that significant amount is taken in waiting for acquiring a lock that is shared with the failover thread. This might be due to, among other things, the increased load on the naming service that this strategy has created as each client refreshes its cache of the active servers from the naming service every 1.5 seconds. When a fault occurs while the background thread is in the process of failing over, then this lock wait time can be observed.
- The client restoration time is lower in this pie chart probably because the relevant servers might have been busier at the time that the 1-client measurements were taken.

As a result of these experiments and analyses, we can state that we have achieved a bounded failover time of one second when the fault detection wait time is set to 1500ms, the fault injection rate is set to 4000ms, at most 10 clients are using the system, two servers are running, the faults are kill-server faults (kill -9 or killServer()), CPU load on the servers is minimal, memory usage on the servers is minimal, no network faults or performance degradations are taking place, the replication manager is running, and all other conditions are normal.