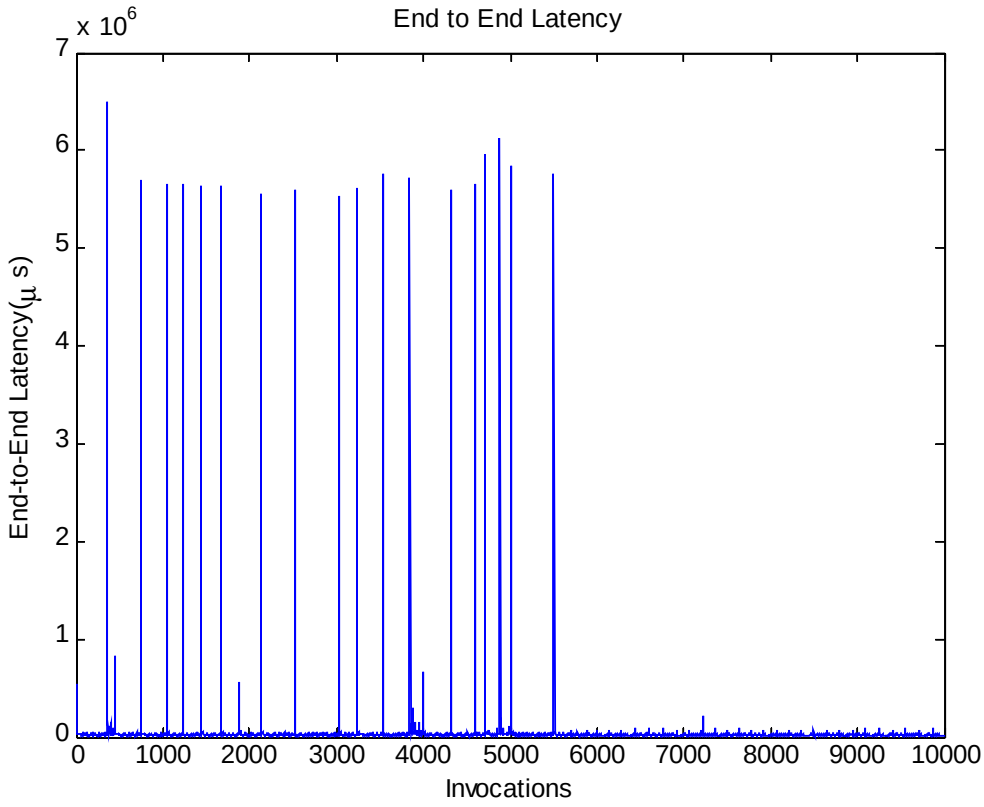
Team 5, Spring 2006

Real – Time Evaluation

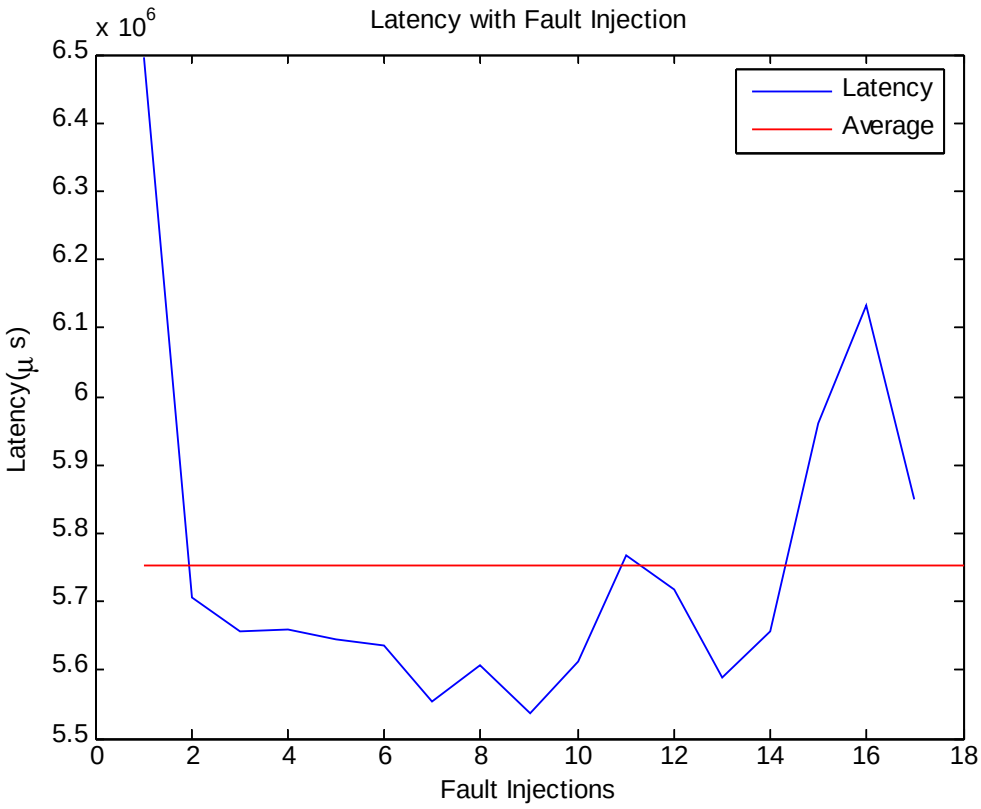
Measurements

Average latency for fault free invocations: 20.55ms
Average latency for only faulty invocations: 5751.9ms
Minimum jitter in the presence of faults: 16.64ms
Maximum jitter in the presence of faults: 742.975ms

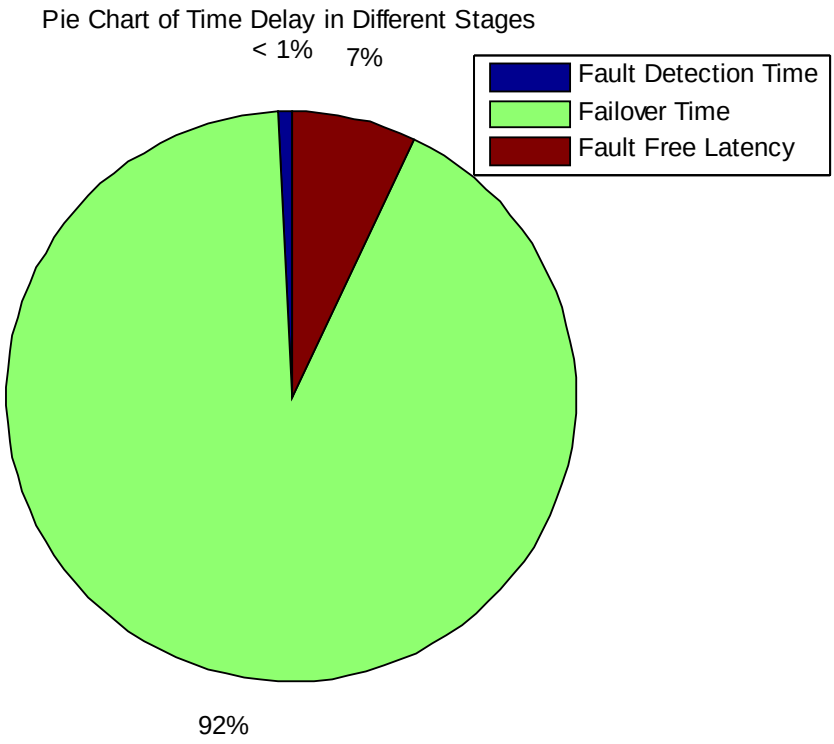
Latency



Jitter


Latency with Fault Injection
x 10^6
6.5
6.4
6.3
6.2
6.1
6
5.9
5.8
5.7
5.6
5.5
Latency(μ s)
0
2
4
6
8
10
12
14
16
18
Fault Injections
Latency
Average

Latency Breakdown



As part of its initialization process, the client will ask the replication manager for the current primary server and create a bean pointing to this server. In a fault-free run, the client will use this same bean for the duration of its run-time. However, in the event that a client invocation fails, the client will have to create a new bean, which points to a working server.

In our system, a faulty invocation is made up of three stages. The first stage, which we are calling fault detection in the pie chart, is that stage where the client attempts to invoke a bean on the primary server but catches an exception indicating that the primary server has gone down. The second stage, called failover in the pie chart above, is that period of time in which the client is attempting to create a new bean which points to the next available backup server. The first step in this process has the client telling the replication manager to remove the primary server from the replication manager's list of servers. Then the client asks the replication manager for the next available backup server, and attempts to create a bean pointing to this new server. The client repeats this process until it successfully creates a bean. The third stage of a faulty invocation, which we are calling fault-free latency in the chart above, is the latency of the successful invocation on the newly created bean.

Initial Thoughts on High-Performance

Our real-time evaluation showed us that failover from the primary server to the backup server made up over 90% of our recovery time for faulty invocations. As a result, our goal in Phase IV will be to improve this failover time by optimizing the failover process.

One bottleneck in our failover mechanism is the replication manager, which takes a considerable amount of time to update its list. The client spends too much time waiting on the replication manager to provide a new server name. The other bottleneck in our system is the process of creating a new bean once the replication manager has provided a valid server name.

Both of these delays can be greatly reduced by always having a second bean ready on the client. When the client starts, it can ask the replication manager for the name of the next backup server along with the name of the primary server. The client can then create two beans - each pointing to these two different servers. In the event that the client cannot invoke a method on the primary server, it can immediately begin using the secondary bean. It can then continue processing as usual and in the background it can get the name of the next backup server from the replication manager and create a new secondary bean. Using this approach, the client will always have a secondary bean readily available in the event that the primary server goes down. This of course assumes that the backup server will not go down before the primary, but if this does happen, the delay would be no worse than in our current setup. Using this approach of always having two beans readily available on the client, we can significanly reduce the end-to-end latency in the presence of primary server failure.