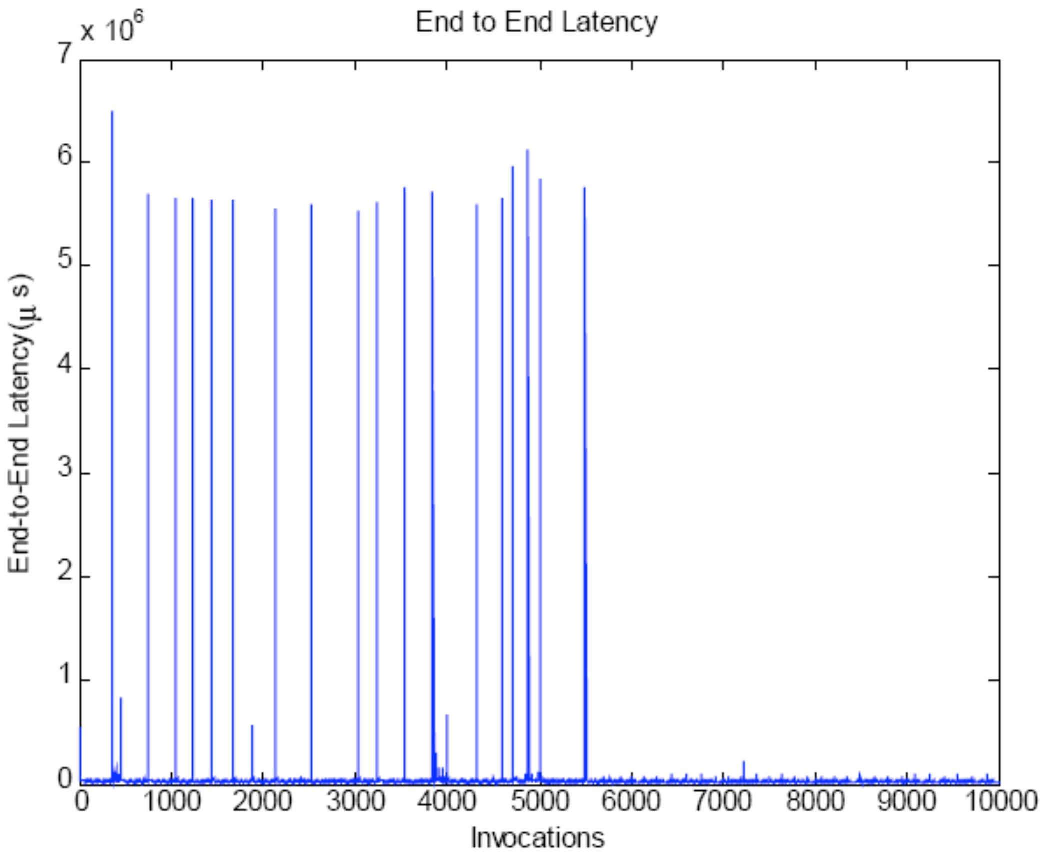
Team 5, Spring 2006

Real-Time Fault-Tolerant Baseline Evaluation

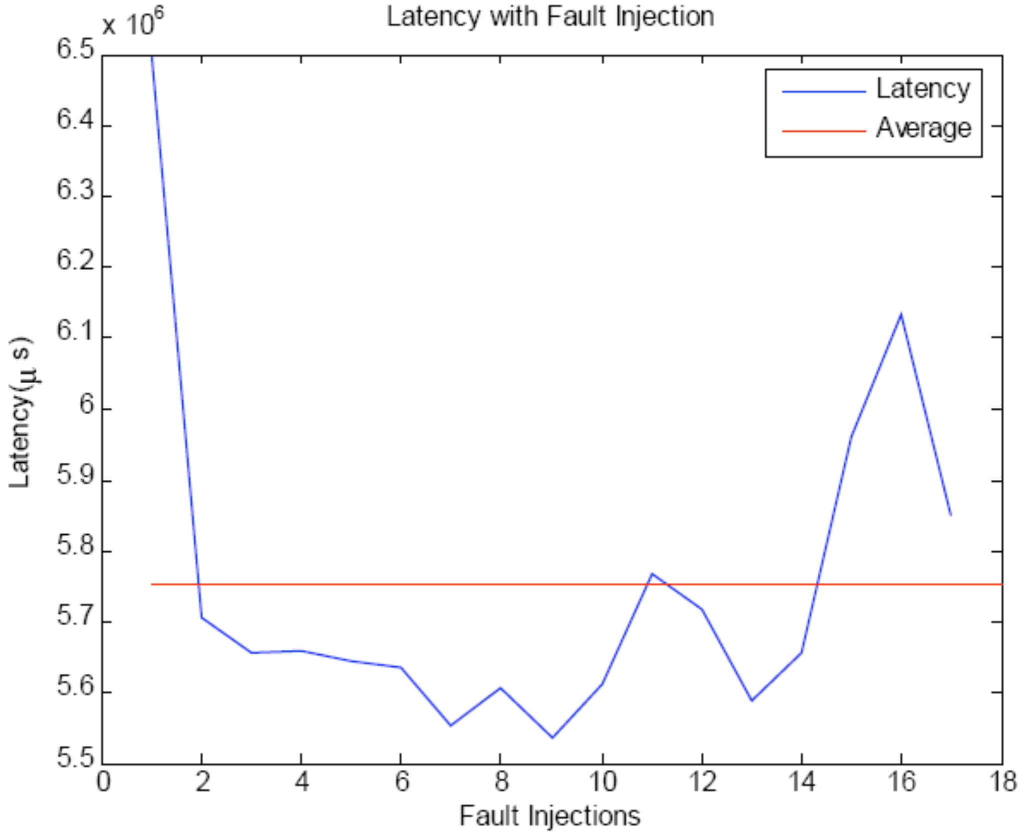
Measurements

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

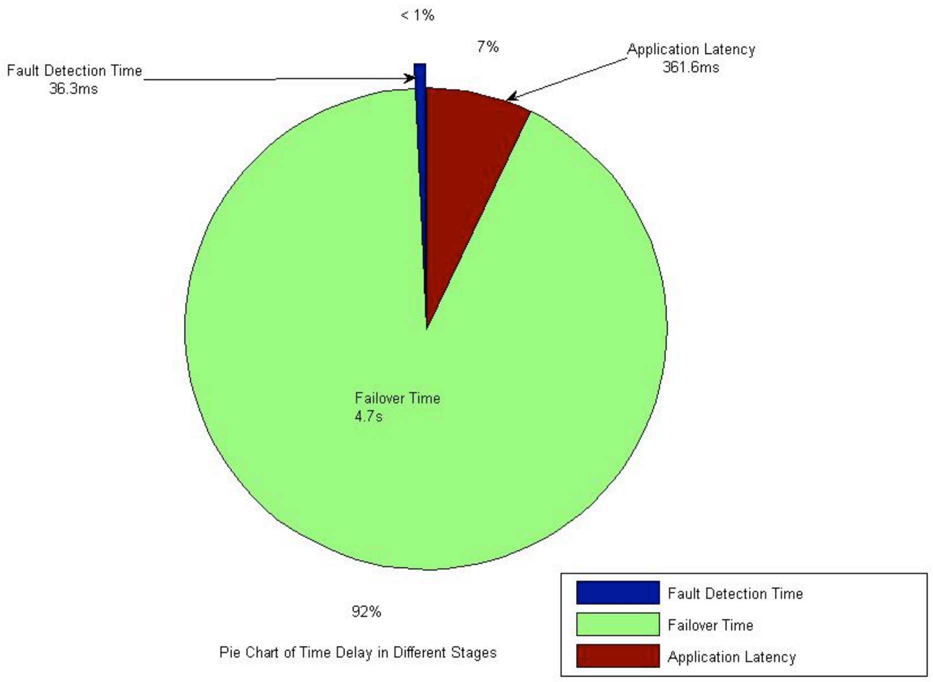
Latency



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

Pie Chart of Time Delay in Different Stages



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 for Implementing 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.

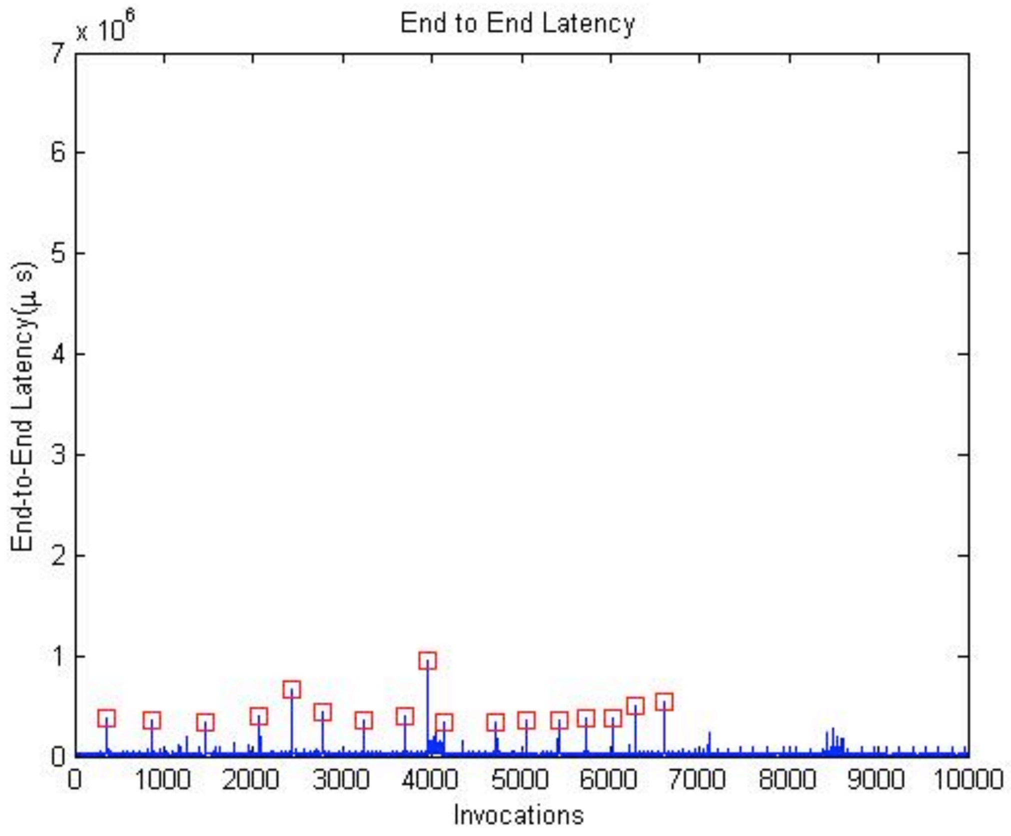
Both of these delays can be greatly reduced by always having a second bean ready on the client. When the client starts, it can ask the replication manager for the name of the next backup server along with the name of the primary server. The client can then create two beans – each pointing to these two 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 significantly reduce the end-to-end latency in the presence of primary server failure.

High-Performance Real-Time Fault-Tolerant Evaluation

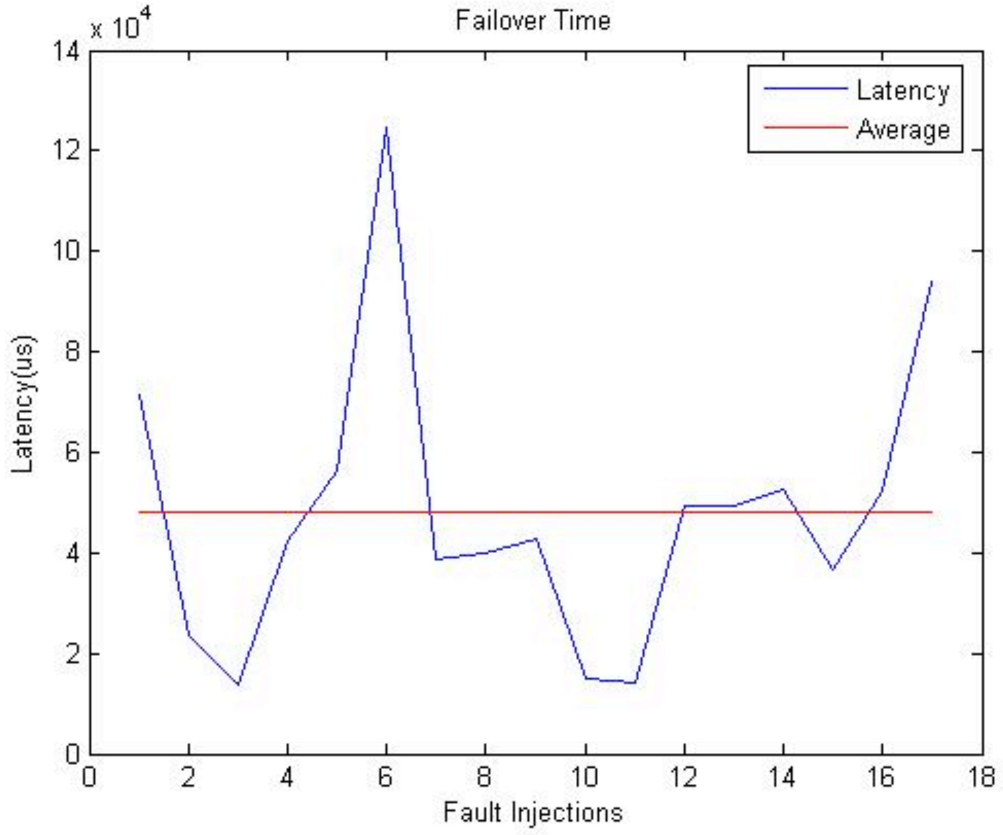
Measurements

Average latency for fault free invocations: 21.30ms
Average latency for only fault invocations: 440.67ms
Minimum jitter in the presence of faults: 3.4ms
Maximum jitter in the presence of faults: 520.3ms

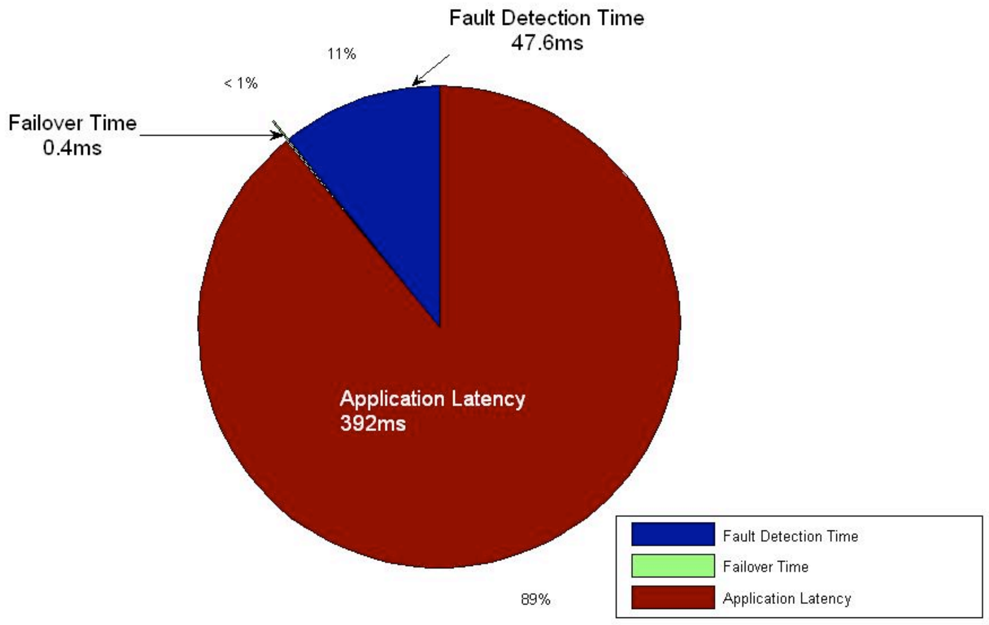
Latency



Jitter


Failover Time
x 10^4
Latency
Average
Latency(us)
Fault Injections
0
2
4
6
8
10
12
14
16
18

Latency Breakdown



Pie Chart of Time Delay in Different Stages

Performance Comparison

In Phase III, failover from the primary server to the backup server made up over 90% of recovery time for faulty invocations. On average, this failover took 4.7 seconds. The failover mechanism applied here involves the client continuously communicating with the replication manager to tell it to update the list of servers. Until the replication manager updates its list, and until the client is able to establish a connection with the new primary server, the client will be stuck in a loop. In this failover mechanism, the bottlenecks come from different aspects:

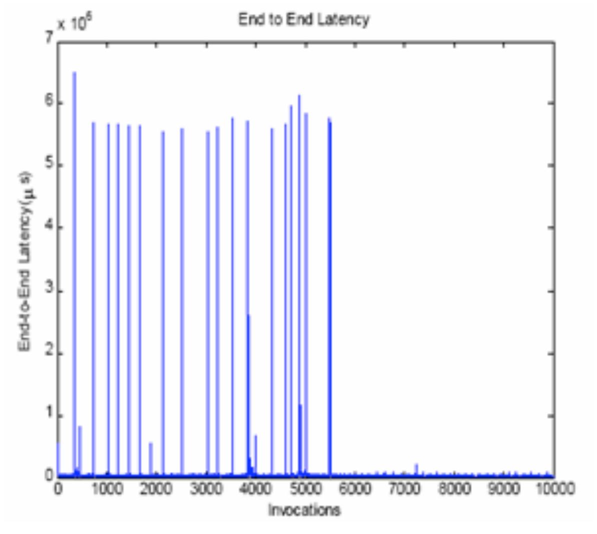
1. Replication manager takes considerable amount of time to update its list.
2. Client spends too much time waiting on replication manager to provide new server name.
3. Client also spends considerable time creating new remote object reference after replication manager has provided valid server name.

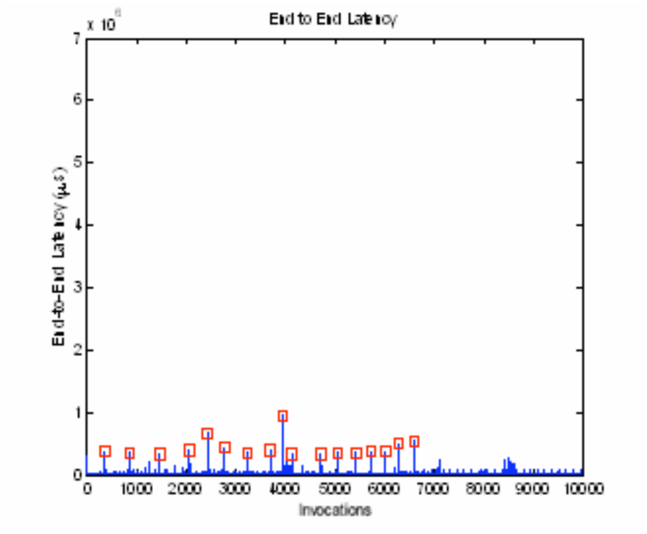
In the Phase IV design, a multi-threaded client has been applied in the new failover mechanism. When the client starts, it asks the replication manager for the name of the primary server as well as for the first backup server, and the client then creates two references pointing to these two different servers. In the event that the client cannot invoke a method on the primary server, it immediately begins using the backup reference and continues processing.

In a separate thread, the client communicates with the replication manager, which will ensure the client has the newest list from the replication manager and its backup reference always points to a new valid backup server. The simulation results showed that the new strategy, which is always having the second remote object reference available to the client, greatly reduced the latency for the bottlenecks seen in Phase III.

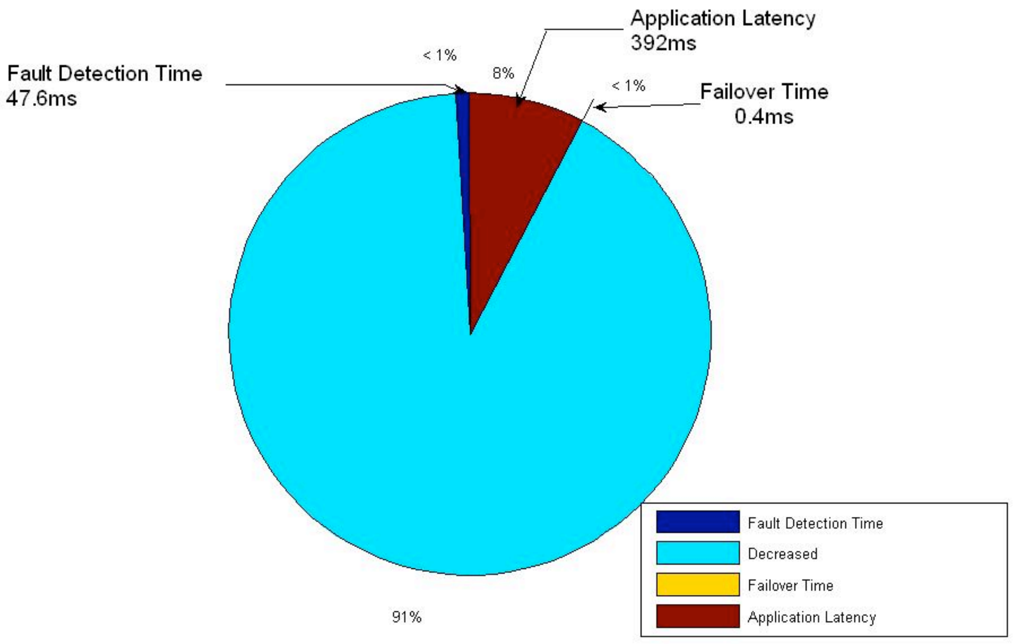
The improvements can be clearly seen when viewing the latency graphs for Phase III and Phase IV.

Latency Comparison





Latency Breakdown



Pie Chart of Time Delay in Different Stages

As seen in the pie chart above, the end-to-end latency for faulty invocations decreased by 91%. The failover time, which is the main component of the latency in phase III, decreased to 0.4ms from 4.7s.

Summary

Latency was decreased by an order of magnitude after the new failover mechanism was applied in the high performance phase. This proved that our analysis of the high latency on failover in the fault tolerant baseline phase was correct. The main component of the latency is now in the server application, which could be shortened by optimizing the application code.

Discussion

1. What happens to the bounded failover times if we increase the load from 1 to x clients?
   In our system, the application latency will increase if we increase the load from 1 to multiple clients. But the failover time and the fault detection time will stay roughly the same, unless the number of clients is large enough to bring heavy loads on the communication channel, CPU and memories.

2. What are the different responses of your system when different kinds of faults, e.g. machine crashed, communication channel breakdown, occur.
   In our system, the same action will be taken if the clients receive any exceptions, which include RemoteException, EJBException, and CreateException. This means that, no matter what kind of fault happens, the client will kill the faulty

primary server and switch to the backup ones. As a result, there is no difference when different faults happen.

3. Is it necessary to set timeout in system? If so, how to set it?
   In our system, the first invocation takes longer than subsequent ones. This happens because we have inter-connections among beans on the application server. During the first invocation, entry bean will call initial context to establish/initialize a connection to its local JNDI service for an interface of the database bean. Thus, in our system, timeout has to be set high enough to avoid hashing the first invocation. In fact, we found that such a high timeout is not necessary to be applied in our system. Nevertheless, if we have to set a timeout, it would have to be longer than the latency of the first invocation.