Jack Davey

**Design Document for Basic Availability**

This document sets out the design for the basic availability functionality that I will implement to complete the second phase of the project.

**Simulation Parameters**

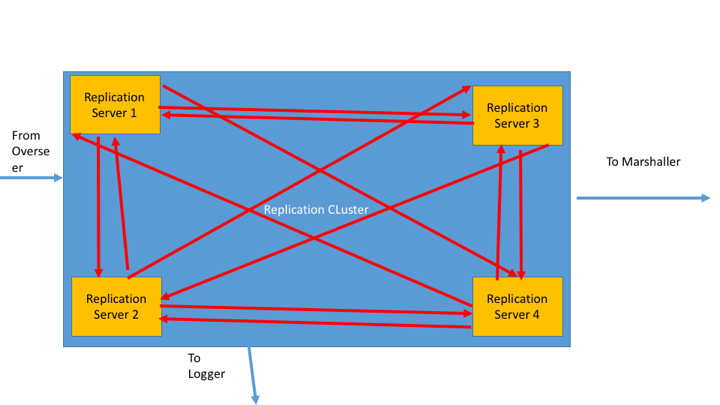
Unlike with the eventual consistency that I previously implemented, there is a lot more scope for customization with basic availability. Because of this, I plan to move all these parameters into their own object. When a system component needs one of these parameters, they will access the value through accessor methods. Mutator methods will be package private, so that they can only be accessed from the user facing code.

**Main Architecture**

The overall architecture of the application will remain mostly unchanged.

The only difference will be the addition of an Avalibility-Checker process that is responsible fo receiving success or failure messages from the replication servers and producing statistics from these. This statistical information is stored in a singleton object that can be accessed from the user facing code. The updated process network is below.

The major work for this piece of functionality will be replacing the individual replication servers with replication clusters. In the eyes of the replication overseer and the replication marshaller, nothing will have changed. These processes still send and receive exactly the same messages to the clusters as they did the original replication servers. Internally, however, each cluster stores several servers adapted from the original replication server.



Each server has one Master node. This is the node that deals with all of the same work as the original replication servers. The master also maintains a list of several slave servers. These servers are responsible for keeping track of all the same data as the master does. The master node preforms the same duties as the original replication server.

Every so often, each server pings the master, if a master has not been heard from for a preset period, then all the slave servers elect a new master from amongst themselves and the old master is relegated to being a slave. all the servers vote by exchanging messages with each other to decide upon the new master.

The main replication cluster handles the organization of master and slave creation and destruction and also handles the election of new masters.

When a database update reches the replication cluster it sends it along to all of the child nodes. When an update reaches a slave in this manner, it randomly decides to accept the update. This is done through the use of random numbers. If the value is accepted, then it is stored in the slave. if not, then it is rejected. IF an update is rejected, than the server that was unable to process it is then considered ‘down’ and is unable to process any more requests for a customizable period of time