Every server, whether a replica or failure detector, pings every failure detection service once a second. When a machine wants to ask whether or not something has failed, it queries each failure detector, which votes that it has failed if the last seen ping occurred more than five seconds ago. The querying machine then takes a majority of the decisions. A primary queries for the status of each of its backups before initiating a backup. A client queries for the status of its primary before sending a request message.

Since our solution relies on pings, we have chosen times of sending pings every second, and deciding failed jvms if the time from last ping exceeds five seconds. Since all transactions begin from a GUI, these won’t be happening a particularly high rate; we believe one second is more than sufficient. Further, we felt that lost pings are a considerable concern; thus, we vote that a jvm has failed only if multiple concurrent pings are never received.

A case that will lead to a halt in the network is if n/2 nodes have failed, where n is the number of jvms initially launched. Since a client has no reason to ask the failure detection service if anything but its branch’s primary have failed, if the primary fails after n/2 nodes have failed, the client cannot reach a majority. Similarly, if a branch does not receive any transactions from its client while n/2 nodes fail, then it also cannot obtain a majority on the next transaction. However, neither of these cases lead to an inconsistent state, and require at least half the network to fail, so we find them acceptable.

In our architecture, there is one failure detection service that all servers query. Each service contains n different failure detection servers, where n is the number of jvms started. Each failure detection server contains n sensors, one for each server. If a sensor does not receive a ping from a server, the failure detection server it is located on is notified the next time it receives a query. If any failure detection server thinks a node has failed, it assumes that everything running on that node’s processor has also failed. Once any replica or client detects a failure, a shutdown message is sent to the jvm suspected of failing to ensure that no one can interact with it erroneously.

We define “work correctly” to mean that we will never reach an inconsistent state with regard to transactions. If a transaction cannot be replicated to all backups, then it is not replicated, and the transaction is considered failed. Only if all replicas in a branch allow a transaction will it occur. If a transaction is in progress and a backup fails, a thread on the primary will detect this failure (by querying the failure detection service) and allow the transaction to terminate.

Our system will work correctly since a majority is needed every time the failure detection service is queried. If a single sensor thinks something has failed, it can continue to vote without affecting the entire service’s decision. Furthermore, once a sensor decides that a jvm is failed, it can never change its mind (even if it gets subsequent pings). This failure decision can only happen in response to a status query from a client or replica.

We assume that colocated on every jvm is a replica of every branch, be it the primary or a backup, and a failure detection server. This allows for the servers and clients to remove a failure detection server from their internal list of failure detection servers when a jvm fails.

clientResolver.txt is assumed to hold host and port for each client. fdsResovler.txt is assumed to hold a port and host for each failure detection server. jvmInfo.txt is assumed to hold a list of replicas to be started by every JVM. jvmResolver.txt is assumed to hold host and port information for each JVM. replicaResolver.txt is assumed to hold a host and port for each replica. topology.txt is assumed to designate a bi-directional link between every pair of branches.

For this project, we created a JVM class to start the threads we want running in a given Java virtual machine. A JVM will automatically start a failure detection server and a series of replicas.

Extra credit i:

Our implementation allows for the safe removal of failed failure detection servers. When a client or replica is notified that a jvm has failed, it removes the failure detection server on that jvm from its internal list of jvms. In case the failure detection was erroneous, a shutdown message is sent to the jvm suspected of failing. The replica or server now has a smaller list of failure detection servers, so a smaller number of votes will constitute a majority. This allows the network to tolerate failures of more than half of the failure detection servers in some cases. In other cases however, failure of more than half of the failure detection servers will be fatal.

Decision to close sockets:

We made a design decision for this project to close sockets after sending messages. We feel that this is a reasonable decision because leaving sockets open consumes system resources. From an implementation perspective, leaving sockets open allows for a much simpler solution. Furthermore, our messaging class is based on our solution for project 1, in which we were already penalized for closing sockets, so we hope you will be lenient ☺.