For our implementation of a fault-tolerant, replicated distributed system, we decided to go with an AP model using a primary/backup approach to attain certain functionality found in CP systems. We implemented this using Python and the Flask library. We liked the idea of having one node that the client talks to, the primary node, and then broadcasting that message to the other nodes, the backups. If it's the case that the client connects to a backup and issues PUT's, GET's, or DELETE's, the backup forwards that message to the primary and then the primary, obviously, broadcasts that message to the rest of the backups. What we found out, however, is that when this forwarding happens, the node that forwarded the message gets hung up and the system freezes.

What we found was that the message and the heartbeats, explained below, would cause the node to wait for a response message from another message that was waiting for a response message from the first message causing the system to permanently wait. So instead of redesigning our heartbeat, we would have the primary broadcast right after the heartbeat was established, or not, as in node failure, so no interference would occur. Another solution was to send the primary the address of the backup forwarding the request and when the primary sends out the broadcast, to skip that node and have the backup perform the request locally. While this worked, we felt this would violate the idea of having primary/backup since the backup would act independently and not by requests from the primary. Don't want any free-thinkers and wild spirits in our system, THERE MUST BE ORDER! =D

Leader election of primary is based on the ordered list of IP's and Ports provided from system environment variables. When the system first starts, the first node on the list will be elected primary, and all other nodes will route any requests they receive through that node. Every

node in the system has a heartbeat thread which checks every 2 seconds if all of the other known nodes in the system are alive. If any node's heartbeat function does not hear back from another node on the system, that node is moved from the alive list to the potentially dead (afk) list. The node that originally sent out the heartbeat then sends out another pulse to all known nodes, asking if they've heard from the proposed dead node. If a single node replies with a 'yes', then the proposed dead node is moved back to the alive list and the system continues as normal. If all nodes agree that the proposed node is in fact dead, the ip is moved to the 'dead' list on all nodes. If the dead node was the primary, a new primary is elected based on the next available node in the sorted 'alive' list. While this isn't optimal for consistency, this makes our system very available, regardless of node deaths or partitions.