Fault Tolerant Distributed Key Value Store

Design

Key-allocation:

Distribution of keys in our system is accomplished by SHA hash that generates a number bounded between 0 and 512 for each key that is put to a server(upper bound set low for debugging). The server will then store this key in it's kvs before broadcasting to the other servers in it's chunk. Servers within the distributed system are 'chunked' into groups based on the total number of available systems and the required number of systems per key-partition. For example, if the system has a requirement of 2 servers per partition and 4 total servers, two of these servers will accept keys that hash from 0-256, then the remaining two servers will accept keys that hash from 257-512.

Updating the view:

Updating the view is accomplished by a single node starting the update process (we will call this node the primary receiver). We instill a minimal amount of authority in the primary receiver. Upon receipt of an update view on the correct route, the client initiates the following procedures:

1. The receiver adds/removes the requested node to its internal state, and re-assigns partitions/chunks as needed via chunk_assign, allocating a new upper and lower bound to each.
2. The primary receiver then sends the new state of the world to all other nodes via broadcast, these receivers will be referred to as 'secondary receivers'.
3. Additionally the primary receiver then sends out another request to all other nodes to begin broadcasting their keys, key by key to the new group to which the keys are assigned.
4. At each secondary receiver the nodes state is set to the state of the incoming broadcast, and it re-initializes its view of the organization of the system.
5. After all servers have broadcasted the required keys to their correct chunks, an accessory function is called: 'prune_db' which iterates the dictionary, and removes any keys which do not belong on the server. This is called via a message from the primary_receiver to all nodes in the network after a small period of latency

Notes:

- On Test 4 we're having an issue where our Django database locks up when it gets spammed with PUTs/GETs as we're using an SQLite database which typically can't handle heavy loads.