Nick Byman and Joe Stein
Capstone Abstract

Features in a Distributed Hash Table

For our distributed systems class we implemented a distributed hash table called Tapestry and used it to build a file system similar to Oceanstore, which Brown has dubbed "Puddlestore." For our capstone project we added hash salting, path caching, and erasure coding to Tapestry.

Before hash salting to look up a key in tapestry we would create a hash of the key and use that hash to look up the relevant node. The issue with this process is that the node that the hash resolves to might be down. To remedy this we add salts to the hashes (we used three different constant strings to append to our pre-hash key) and if the node that the hash resolves to is down we look up a hash with a different salt.

With path caching, if node A publishes an object X to root R, all nodes along the path cache the fact that A has X. It can then respond to any requests for X itself. As with the root, cached entries expire slightly after one republish period after they are created. The motivation for path caching is that lookups are faster because the lookups can terminate earlier (i.e., a given node can terminate the lookup early when it knows where an object is).

While hash salting addresses *root* failure, failure of object-storing nodes remains problematic. Publishing entire objects at multiple nodes is one solution to this issue, but the amount of storage space required would be rather large. We implemented erasure coding to store the XOR of each data block in tapestry. The encoder takes a block, splits it in two, XORS the two blocks to create a parity block, and stores all three blocks in tapestry. We can recover all three of these blocks from any two of them.