DHT Report

RogerW

July 15, 2025

1 Design of Chord

Most of the design is same as the original Chord paper, with some modifications to improve performance and reliability.

1.1 Active replica

To ensure replica consistency and availability, instead of statically assigning replicas to specific nodes, we let the leading replica (defined as the node that is the direct successor of the data key) to actively publish its data. Replicas that the node is not responsible for and doesn't receive the publish will be cleared after expire time.

1.2 Implementing Delete

Rather than trying to delete all replicas, we just need to delete on enough nodes, so that the nodes with the staled replica won't be the leading one before expire time.

1.3 Clear upon failure

To ensure routing availability when node quit, we ensures that a failure in accessing the node will cause it to be cleared out from the routing table and successor list, promising that after finite amount of retrying, the routing will eventually finish.

1.4 Retry as error handling

Instead of doing the best effort, all remote calls will immediately fail when any error occurs, and it will start retrying after a exponential backoff. By the above clear mechanism, we can ensure that each retry will be effective.

2 Design of Kademlia

Our design implements the core Kademlia protocol but enhances it with several mechanisms for improved robustness and data management, which are not standard features.

2.1 Logical Deletion with Tombstones

The standard Kademlia does not define DELETE. We implement deletion using tombstones. A Delete operation does not erase data; instead, it publishes a new version of the value with an IsTomb flag set to true and a fresh timestamp. This tombstone propagates through the network like any other data and overrides older, valid entries due to the "last-write-wins" timestamp policy.

2.2 Repair when quit

When a node leaves gracefully, it performs two key actions to maintain network health:

- **Network Healing:** The node actively repairs the network topology around itself. It does this by:
  1. Identifying its own k closest neighbors.
  2. Iterating through this list of neighbors, pairing them up (A and B, B and C, etc., forming a ring).
  3. For each pair, it makes a remote call to neighbor A instructing it to add neighbor B to its routing table, and vice-versa. This forges direct connections between the departing node's peers, strengthening local connectivity and patching the hole left by its absence.
- **Broadcast Notification:** Finally, the node iterates through every unique peer in its k-buckets and sends them a NotifyLeave message in parallel. This allows all aware nodes to immediately and cleanly prune the departing node from their routing tables, preventing future failed lookups.

2.3 Quorum-Based success

Our Publish function (used by Put and Delete) requires to return success. An operation is only considered successful if it receives positive acknowledgment from a majority (k/2) of the target nodes.

2.4 Retry with blacklist

Unlike Chord, which retry is ensured to make progress, Kademlia cannot remove the failed node from the source which returns it easily. Instead, we maintain a blacklist of failed nodes for a certain period. If a node fails, it is added to this blacklist, and subsequent operations will avoid contacting it.