COP5615

Chord Protocol

Ankit Soni- 8761-9158

Kumar Kunal- 5100-5964

Bonus Report

The implementation is Consistent hashing where the nodes and messages reside in the same hashing space. The API used in the Chord protocol implementation are: *find_successor, find_predecessor, closest_preceding_finger, join, init_finger_table, update_finger_table, stabilize, notify and fix_fingers*

The failure model was implemented as per section 5 of the research paper. We kill the nodes after they join the chord before message passing happens. What we are trying to achieve is correct finger tables for each node before the actual message passing happens to avoid looking up any dead process. This was verified.

Some observations:

- Enough time for stabilization is required so that the finger tables are updated correctly after the initialization. We achieve this by some delay in each cycle.
- When the nodes are killed, we simply discard the input number of nodes to be discarded. The problem now remains is to make sure all the finger tables have the updated neighbors and that the killed nodes are purged from each node's finger table.
- This is achieved by stabilization in the next steps. From our observations the finger table gets updated correctly 100 percent of the times.
- This means that the system is a 100 percent fault tolerant
- In the context of peer to peer searches, since the search keys are always in the numerical range of the node ids, once some nodes are removed from the network, these keys align themselves with other nodes(ownership), and the lookups work just as effectively. The redistribution of keys to different servers(nodes) is out of the scope of the project as the keys are just numerical values. Here we are mainly focusing on the Chord protocol.
- For extremely large number of nodes, the system takes almost linear time. This is because enough stabilization time for these nodes is essential before the first round of message passing.
- Irrespective of the stabilization times, when the actual message passing happens, the lookups always yield in log n time for any number of nodes, and any number of messages.
- For large number of messages, of the order of the number of nodes in the chord protocol, the average number of hops for a lookup reduces significantly.

- In conclusion this is an extremely resilient and fault tolerant protocol which is easy to maintain with the join and stabilize APIs.