Report : Chordy, an Erlang implementation of DHT based on Chord

Avneesh Vyas

avneesh@kth.se

October 12, 2016

1 Introduction

The report describes the various steps and procedures followed during the implementation of an erlang based Distributed Hash Table (DHT). The DHT in this implementation is based on the Chord scheme. The participating nodes in DHT are arranged in a virtual ring in which nodes can continuously join and leave the system. Joining and leaving of nodes triggers key handover to or from the node.

2 Main features of DHT

A DHT is scalable key-value store that uniformly distributes keys among its participant. A client wishing to store a key-value pair simply requests one of the members of the ring. On receiving the request, the ring using the key finds the node identifier which should store the node. Similarly, for lookup the client can request any node of the ring, and the ring through key finds the node storing the value corresponding to that node.
The main features of an efficient DHT are:
Fault Tolerance: Rings should handle the scenarios where nodes join and leave by keeping track of predecessors and successors
Data Replications: When nodes crashes in the DHT, it is important that ring does not lose the data store of that node.
Uniform Key distribution: Keys should be distributed uniformly so that some nodes do not get overloaded and others under-utilized.

3 Algorithm

The algorithm for managing a DHT is based on keeping track of Predecessor and Successor nodes. Every node is designated to store keys starting from predecessor's Id to its own Id. This way any ring member

that receives a lookup request can forward the request to its successor node if the key does not fall in its range.
To be tolerant to nodes joining and leaving the ring, each node periodically enquires its successor node for its current predecessor. If the node detects that its successor node's predecessor has changed to a new node, then it attempts to make the new node as its successor.
To handle cases of node crashes, each node maintains pointers to multiple successor nodes. In our implementation, 2 consecutive successor nodes' pointers were maintained. So, if the node detects that its immediate successor is not reachable, it can make the 'next' node its successor.
In both of the above cases of node joining or leaving the system, keys on the affected nodes should be re-distributed for efficient routing.

One way to ensure that nodes crashes do not cause loss of data store, data store is replicated onto successor node. This way if a node dies, successor node can take over the responsibility of its predecessor. Of course it does not handle the scenario where two consecutive nodes die simultaneously.

4 Verification

The fault tolerant behavior of a 5-member ring was tested by killing some nodes in the ring and checking if the pointers to predecessor and successor nodes were successfully updated.

5 Conclusion

With this assignments, author was introduced to the design challenges of a distributed hash table. During experimentation, the author observed that 5000-key look up on a single node took 109ms. As opposed to this, the same look up just took 62ms on a 5-member DHT. So, distributed hash table provides an efficient means to maintain large data store with lower look up time.