Distributed Hash Tables

DHTs

A **distributed hash table (DHT)** is a class of a decentralized distributed system that provides a lookup service similar to a **hash table: (key, value)** pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.

Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption.

This allows a DHT to **scale** to extremely large numbers of nodes and to handle continual node arrivals, departures, and **failures**.

Where do we find DHTs?

In multitudes of distributed data storage/retrieval systems. Some examples

- Database Service -distributed NoSQL key-value data stores:

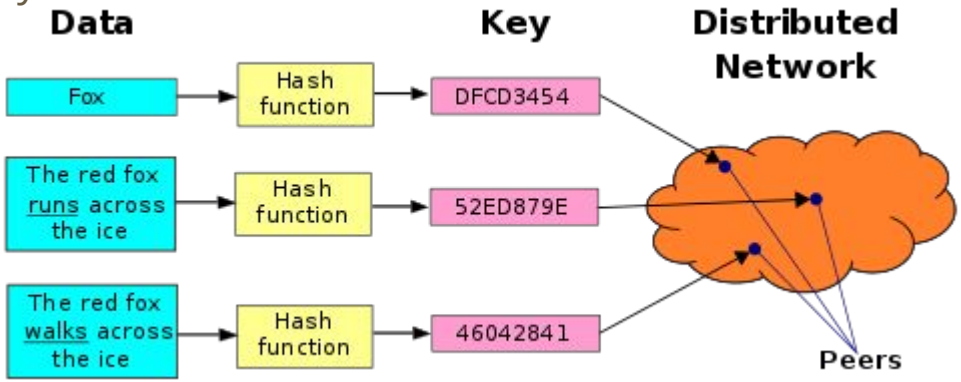
  Apache Cassandra,

  Riak,

  Voldemort, (used by LinkedIn)

  Amazon DynamoDB,

- AKAMAI CDN
- BitTorrent - Magnetic links

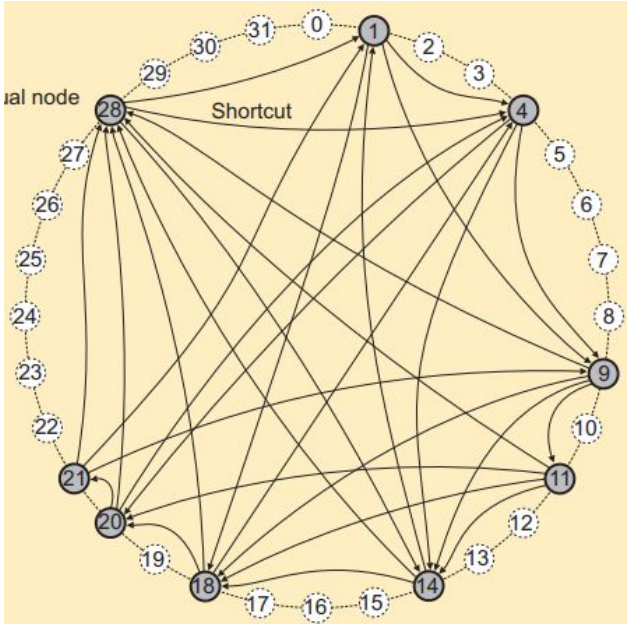


Chord Protocol

Chord is one of the four original distributed hash table protocols, along with **CAN, Tapestry, and Pastry**.

Simplistic method of checking immediate neighbours is a **non** solution

10,000 nodes ring => worst case 5000 hops?



Chord Protocol

A lookup generally requires **O(log N)** step, with N being the number of nodes in the system.

In a large DS, nodes will often join, leave, fail...

Joining a DHT system:

‘p’ wants to join -> lookup for succ(p+1) -> insert itself before succ(p+1)

Relevant *files are transferred to p* and *finger tables updated*

Leaving is as similar...

Joining, leaving the network

Each node keeps track of its **predecessor** too

Keeping the finger table up-to-date:

> Each node q regularly does this: contacts succ(q+1) and requests to return pred(succ(q+1))
>
> If q = pred(succ(q+1)), then consistent; otherwise update succ(q+1) i.e, adjust $FT_q[1]$
>
> Subsequently update the remaining entries of FT by resolving succ(...)

Chord usually does these updates as background process

Likewise, each node will check if its predecessor is alive [Stoica et al., 2003]