Distributed Hash Tables

Material derived from slides by I. Gupta, M. Harandi, J. Hou, S. Mitra, K. Nahrstedt, N. Vaidya
Distributed System Organization

• Centralized
• Ring
• Clique

• How well do these work with 1M+ nodes?
Centralized

• Problems?
• Leader a bottleneck
  • $O(N)$ load on leader
• Leader election expensive
Ring

- Problems?
- Fragile
  - $O(1)$ failures tolerated
- Slow communication
  - $O(N)$ messages
Clique

• Problems?
• High overhead
  • $O(N)$ state at each node
  • $O(N^2)$ messages for failure detection
Distributed Hash Tables

• Middle point between ring and clique

• Scalable *and* fault-tolerant
  • Maintain $O(\log N)$ state
  • Routing complexity $O(\log N)$
  • Tolerate $O(N)$ failures

• Other possibilities:
  • State: $O(1)$, routing: $O(\log N)$
  • State: $O(\log N)$, routing: $O(\log N / \log \log N)$
  • State: $O(\sqrt{N})$, routing: $O(1)$
Distributed Hash Table

- A hash table allows you to insert, lookup and delete objects with keys

- A distributed hash table allows you to do the same in a distributed setting (objects=files)

- DHT also sometimes called a key-value store when used within a cloud

- Performance Concerns:
  - Load balancing
  - Fault-tolerance
  - Efficiency of lookups and inserts
Chord

- Intelligent choice of neighbors to reduce latency and message cost of routing (lookups/inserts)

- Uses *Consistent Hashing* on node’s (peer’s) address
  - 
  - (ip_address, port) $\rightarrow$ hashed id ($m$ bits)
  - Called peer id (number between 0 and $2^m - 1$)
  - Not unique but id conflicts very unlikely
  - Can then map peers to one of $2^m$ logical points on a circle
Ring of peers

Say $m=7$

6 nodes
Peer pointers (1): *successors*

Say $m=7$

(similarly predecessors)
Peer pointers (2): *finger tables*

Finger Table at N80

<table>
<thead>
<tr>
<th>i</th>
<th>ft[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>5</td>
<td>112</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
</tbody>
</table>

Say \( m = 7 \)

\[ \text{ith entry at peer with id } n \text{ is first peer with id } \geq n + 2^i (\text{mod } 2^m) \]
Mapping Values

- Key = hash(ident)
  - m bit string
- Value is stored at first peer with id greater than its key (mod $2^m$)
Search

Say $m=7$

Who has cnn.com/index.html (hashes to K42)

File cnn.com/index.html with key K42 stored here
Search

At node $n$, send query for key $k$ to largest successor/finger entry $\leq k$
if none exist, send query to $\text{successor}(n)$

Say $m=7$

Who has $\text{cnn.com/index.html}$?
(hashes to $K42$)

File $\text{cnn.com/index.html}$ with key $K42$ stored here
Search

At node $n$, send query for key $k$ to largest successor/finger entry $\leq k$
if none exist, send query to $\text{successor}(n)$

Say $m=7$

File cnn.com/index.html with key K42 stored here

Who has cnn.com/index.html?
(hashes to K42)

All “arrows” are RPCs
Analysis

Search takes $O(\log(N))$ time

**Proof**

- (intuition): *at each step, distance between query and peer-with-file reduces by a factor of at least 2* (why?)
  Takes at most $m$ steps: is at most a constant multiplicative factor above $N$, lookup is $O(\log(N))$

- (intuition): after $\log(N)$ forwardings, distance to key is at most $2^m / N$ (why?)
  Number of node identifiers in a range of $\frac{2^m}{N}$ is $O(\log(N))$ with high probability (why?)
  So using *successors* in that range will be ok
Analysis (contd.)

• $O(\log(N))$ search time holds for file insertions too (in general for routing to any key)
  • “Routing” can thus be used as a building block for
    • All operations: insert, lookup, delete

• $O(\log(N))$ time true only if finger and successor entries correct

• When might these entries be wrong?
  • When you have failures
Search under peer failures

Say $m=7$

Who has `cnn.com/index.html` (hashes to K42)

Lookup fails (N16 does not know N45)

File `cnn.com/index.html` with key K42 stored here
Search under peer failures

One solution: maintain $r$ multiple successor entries. In case of failure, use successor entries.

Say $m=7$

Who has \texttt{cnn.com/index.html}?

(Hashes to $K42$)

File \texttt{cnn.com/index.html} with key $K42$ stored here
Search under peer failures (2)

Say $m=7$

Who has ${cnn.com/index.html}$?
(hashes to $K_{42}$)

File $cnn.com/index.html$ with key $K_{42}$ stored here

Lookup fails (N45 is dead)
Search under peer failures (2)

One solution: replicate file/key at \( r \) successors and predecessors

Say \( m=7 \)

Who has \( \text{cnn.com/index.html} \)? (hashes to \( K42 \))

File \( \text{cnn.com/index.html} \) with key \( K42 \) stored here
Need to deal with dynamic changes

- Peers fail
- New peers join
- Peers leave
  - P2P systems have a high rate of *churn* (node join, leave and failure)

→ Need to update *successors* and *fingers*, and copy keys
New peers joining

Introducer directs N40 to N45 (and N32)
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it

Say $m=7$
New peers joining

Introducer directs N40 to N45 (and N32)
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it
\textit{N40 periodically talks to its neighbors to update finger table}

Say $m=7$

Stabilization Protocol (to allow for “continuous” churn, multiple changes)
New peers joining (2)

N40 may need to copy some files/keys from N45
(files with fileid between 32 and 40)

Say $m=7$
Lookups

Average Messages per Lookup

Number of Nodes

log N, as expected
Chord Protocol: Summary

• $O(\log(N))$ memory and lookup costs

• Hashing to distribute filenames uniformly across key/address space

• Allows dynamic addition/deletion of nodes
DHT Deployment

• Many DHT designs
  • Chord, Pastry, Tapestry, Koorde, CAN, Viceroy, Kelips, Kademlia, ...

• Slow adoption in real world
  • Most real-world P2P systems unstructured
    • No guarantees
    • Controlled flooding for routing
  • Kademlia slowly made inroads, now used in many file sharing networks