Distributed Hash Tables

Jasleen Kaur

November 11, 2009

Distributed Hash Table (DHTs)

- Hash table: data structure that maps “keys” to “values”
  - essential building block in content systems

- Distributed Hash Table: similar, but spread across the Internet
  - Each node stores (key, value) pairs
  - Interface:
    - Insert(key, value)
    - Lookup(key)
    - Join/leave
  - Each DHT node in the overlay supports single operation:
    - given input key, route messages toward node holding key

- “Middleware” for building distributed systems
  - DNS, File Systems, ...
DHT In Action

Operation: take key as input; route messages to node holding key

DHT In Action: insert()

Operation: take key as input; route messages to node holding key
### DHT In Action: lookup()

- **Operation**: take key as input; route messages to node holding key.

### DHT Design Goals

- An “overlay” network with:
  - flexible mapping of keys to physical nodes
  - small network diameter
  - small degree
  - local routing decisions
- A “storage” or “memory” mechanism with
  - best-effort persistence (soft state)
- We’ll look at two designs:
  - Chord
  - Pastry
**Chord**

- Based on logical m-bit identifiers
  - 0 to $2^m-1$ ordered in an identifier “circle” (modulo $2^m$)

- (Key, Value) pairs are stored/located by using a **consistent hash** function $CH_k$ to map keys, $K$, onto a point $\Phi$ on the circle
  - $\Phi = CH_k(K)$

- System nodes are also mapped onto points, $N_i$, on the same identifier circle
  - $\Phi$ may be greater than $N_i$

- Node $N_i$ stores all $(K,V)$ pairs where $K$ maps to a point $\Phi$ such that $N_i$ is the first node where
  - $\Phi \leq N_j$ (where $N_j$ is the successor of $\Phi$)
Chord

- DHT API:
  - Each node stores (key, value) pairs
  - Interface:
    - insert(key, value)
    - lookup(key)
    - join/leave
  - Each DHT node in the overlay supports single operation:
    - given input key, route messages toward node holding key
Simple Lookup -- recursive mode
(part two: return successor & send query)

Memory: O(1)
Mean lookup is O(n^2)
Not Scalable!

Scalable Lookup With Small Node State
(part one: use local “finger table”)

Finger table at node j:
for i=0 to log(n)
finger[i] = Successor(Node(j+2^i mod 2^log(n)))
Scalable Lookup With Small Node State
(part two: use remote finger table data)

Scalable Lookup With Small Node State
(part three: locate successor node)

Finger tables help halve the ID-space distance in each step.

Mean lookup is $O((\log n)^2/2)$
With $m$ table entries

$\cdots$ Schnieder
Chord

- **DHT API:**
  - Each node stores (key, value) pairs
  - Interfaces:
     - lookup(key)
     - insert(key, value)
     - Join/Leave
  - Each DHT node in the overlay supports single operation:
    - given input key, route messages toward node holding key

Node Join
(example, Hash(128.250.6.182) = 26)

- Nodes also maintain a **predecessor** link (not used for search)
- (1) Joining node contacts any existing node to find successor
- (2) Successor link created from returned value
Node Join
(example: Hash(128.250.6.182) = 26)

- (3) Successor Notified and data for keys < 26 moved and predecessor link made.
- (4) Periodic Stabilize protocol run by all nodes updates successor link in predecessor node (N21) and predecessor link in new node; Fix Fingers also run to fix finger tables (uses find successor search).

Replication & Robustness:
Each node maintains list of $r$ successors

Protocol against simultaneous node failures that could result in loss of correct successor links.

Applications run replicate data at $k$ of the $r$ successors to provide high availability in event of node failures.
The Chord Theorems

Theorem 12.1: For any set of \( N \) nodes and \( K \) keys, with high probability, the following is true:
1. Each node is responsible for at most \((1 + 1/K)\) keys.
2. When an \( O(N) \) node joins or leaves the network, the responsibility of \( O(K/N) \) keys changes hands (and only to or from the joining or leaving node).

Theorem 12.4: With high probability, the number of nodes that can be contacted in \( O(N/\log N) \) time in an \( N \)-node network is \( O(N/\log N) \).

Theorem 12.3: If any sequence of join operations is encountered interleaved with stabilizations, then at some time after the last join the successor pointer will form a cycle on all the nodes in the network.

The Chord Theorems (cont.)

Theorem 12.2: If we take a stable network with \( N \) nodes with correct finger pointers, and another set of up to \( N \) nodes joins the network, and all successor pointers that perhaps not all finger pointers are correct, then bootstrap will still take \( O(N/\log N) \) time with high probability.

Theorem 12.5: If we use a successor list of length \( s = \Theta(\log N) \) in a network that is initially stable, and does not move, fails with probability \( 1/2 \), then with high probability \( \text{find-successor} \) returns the closest living successor in \( O(N/\log N) \) time.

Theorem 12.6: In a network that is initially stable, if every node then fails with probability \( 1/2 \), then the expected time to execute \( \text{find-neighbor} \) is \( O(N/\log N) \).