COMP 790-088: Networked & Distributed Systems

Distributed Hash Tables

Jasleen Kaur

November 11, 2009

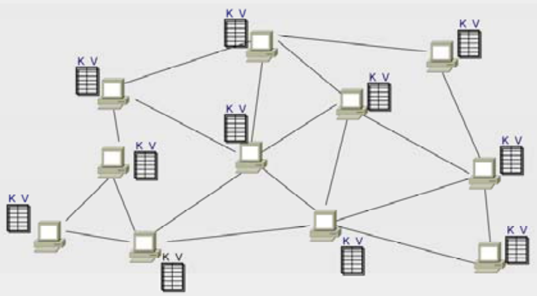
1

Distributed Hash Table (DHTs)

- Hash table: data structure that maps "keys" to "values"
  - » essential building block in software 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, I3, Content distribution, ….

2

DHT In Action

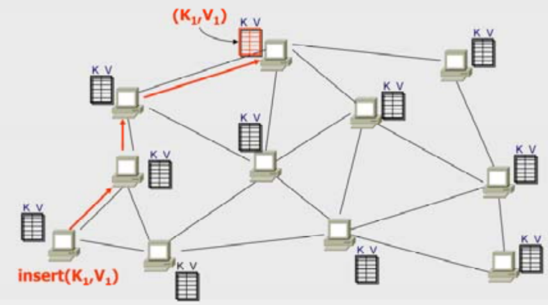


Operation: take *key* as input; route messages to node holding *key*

3

3

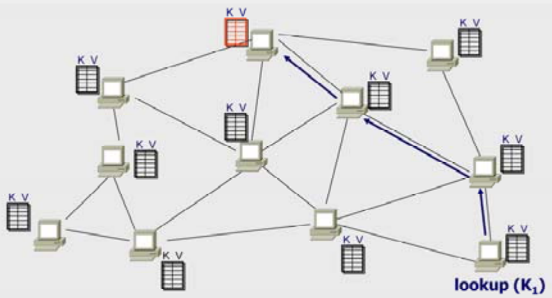
DHT In Action: insert()



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

4

DHT In Action: lookup()



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

5

5

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

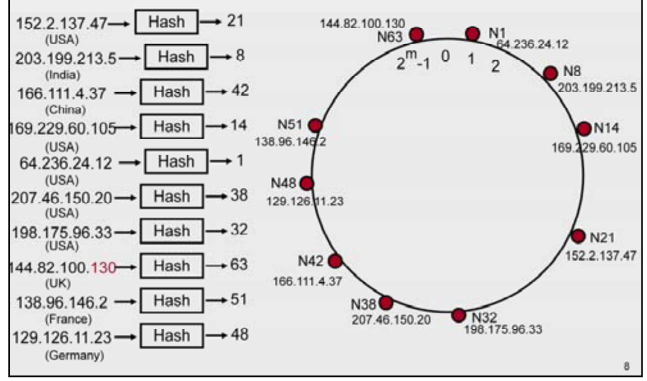
6

6

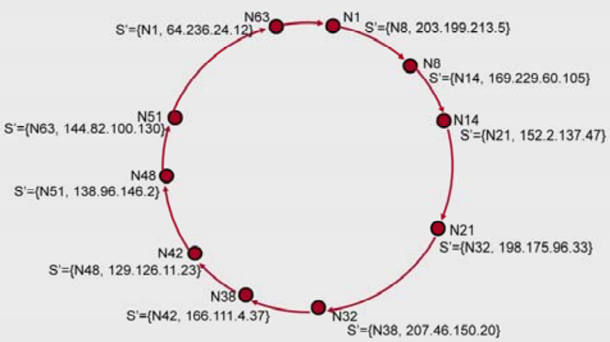
Chord

- 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,*** to map keys, ***K,*** onto a point, Φ, on the circle
  - » $\Phi = CH(K)$
- System nodes are also mapped onto points, N_i, on the same identifier circle
  - » # Key values may be greater than # Nodes
- Node N_i stores all (*K,V*) pairs where *K* maps to a point Φ such that N_i is the <u>first</u> node where
  - » $\Phi \le N_i$ (N_i is the <u>*successor*</u> of Φ)

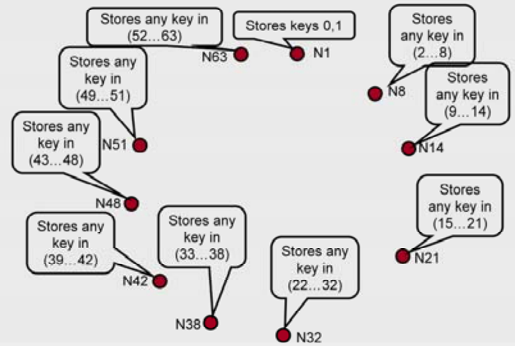
Hash IP address to Node ID (N) (example with m=6)



Nodes Maintain Successor Pointer (S')



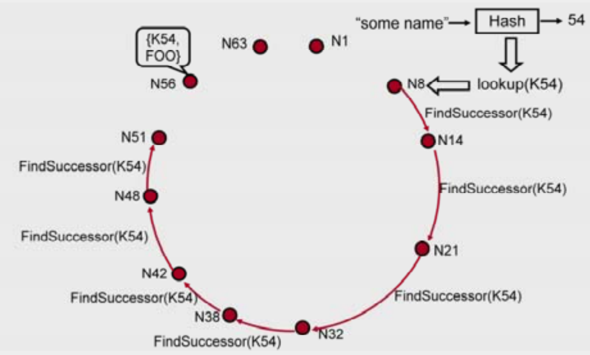
Key locations in example



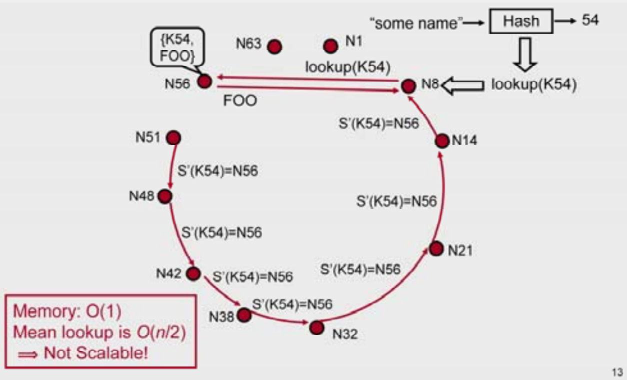
Chord

- 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 one: find successor of key)

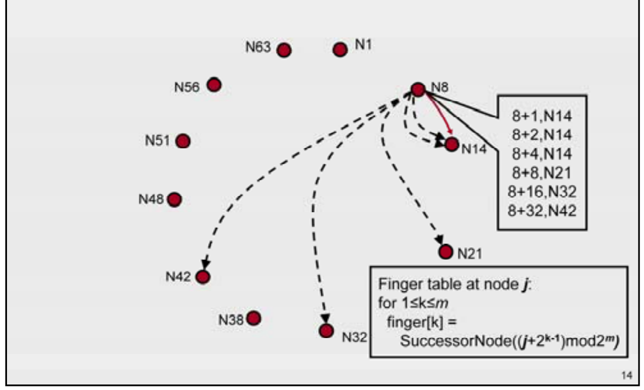


Simple Lookup -- recursive mode (part two: return successor & send query)



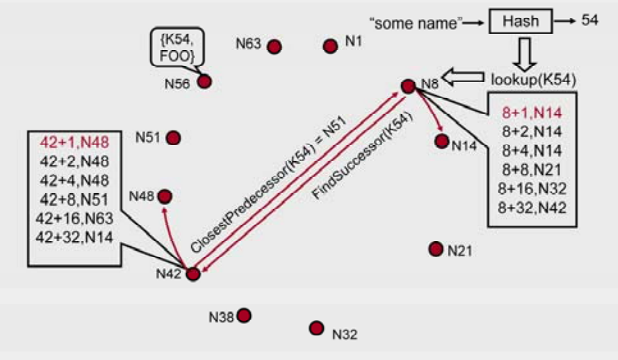
13

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

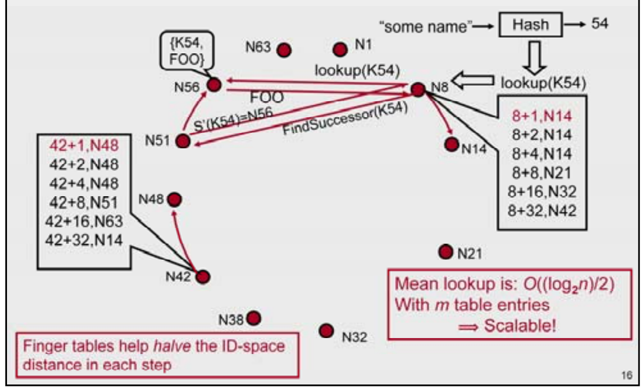


14

Scalable Lookup With Small Node State (part two: use remote finger table data)



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

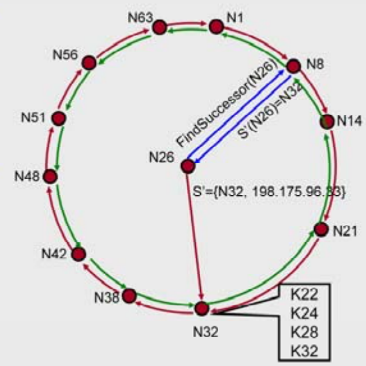


Chord

- DHT API:
  - Each node stores (key, value) pairs
  - Interface:
    - 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

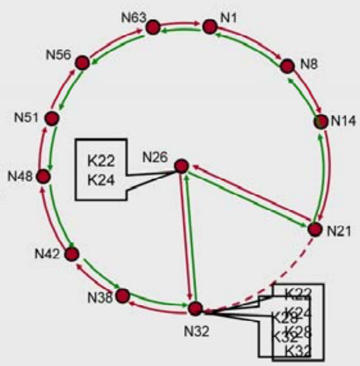
Node Join (example, Hash(128.250.6.182) = 26)

- 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.



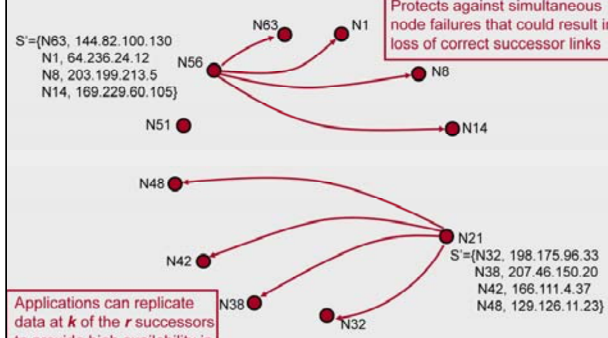
Node Join
(example, Hash(128.250.6.182) = 26)

- (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 FindSuccessor() search)



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

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



Applications can replicate data at ***k*** of the ***r*** successors to provide high availability in event of node failures

The Chord Theorems

Theorem IV.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 $(1+\epsilon)K/N$ keys.
2) When an $(N+1)$th node joins or leaves the network, the responsibility for $O(K/N)$ keys changes hands (and only to or from the joining or leaving node).

Theorem IV.2: With high probability, the number of nodes that must be contacted to find a successor in an N-node network is $O(\log N)$.

Theorem IV.3: If any sequence of join operations is executed interleaved with stabilizations, then at some time after the last join the successor pointers will form a cycle on all the nodes in the network.

The Chord Theorems (cont.)

Theorem IV.4: 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 (but perhaps not all finger pointers) are correct, then lookups will still take $O(\log N)$ time with high probability.

Theorem IV.5: If we use a successor list of length $r = \Omega(\log N)$ in a network that is initially stable, and then every node fails with probability 1/2, then with high probability *find_successor* returns the closest living successor to the query key.

Theorem IV.6: In a network that is initially stable, if every node then fails with probability 1/2, then the expected time to execute *find_successor* is $O(\log N)$.