

Chord: A scalable peer-to-peer lookup protocol for Internet applications

Stoica, et al. IEEE/ACM Trans. On Networking, Feb 2003.

S. Venkatesan

Department of Computer Science

Fall 2007

Chord

- g Problem:
 - Nodes join and leave the network constantly
 - Need to know where certain item is available for lookup or download
- g Given a key, map key to a node that has key

S. Venkatesan

Department of Computer Science

Fall 2007

Main Component

- g Consistent hashing
 - Balance load
 - Each node gets roughly same number of keys
 - Small movement when nodes join/leave
 - Each node knows of small number of other nodes
- g Each node is aware of most nodes in previous works

S. Venkatesan

Department of Computer Science

Fall 2007

Consistent Hashing

- g Each node has info about few nodes.
- g Completely distributed
 - No node is more important than others
- g For a N-node system
 - Each node maintains info about $O(\log N)$ nodes
 - All lookups resolved in $O(\log N)$ messages
- g Nodes join and leave system; it is OK

S. Venkatesan

Department of Computer Science

Fall 2007

Chord's characteristics

- g Load balance
 - Spread keys evenly among nodes
- g Decentralized
 - Fully distributed
- g Scalability
 - Lookup grows as log of # of nodes
- g Availability
 - Nodes can join, leave, fail
 - Node responsible for key is always found
 - Even if system in continuous state of change
- g Flexible naming
 - Flat naming

S. Venkatesan

Department of Computer Science

Fall 2007

Examples of Chord applications

- g Read on page 19

S. Venkatesan

Department of Computer Science

Fall 2007

Model

- g $m = \log(N)$ where N is the maximum number of nodes that there may be
- g Items; each item has a key k ;
- g K is total number of keys;
- g Total number of items is $O(2^m)$.
- g $K \approx N$

Problem

- g Participating nodes store certain number of items (or keys)
- g Lookup: find node that has a given key k
- g New items may be added; items removed
- g New nodes may join, nodes may leave
 - At any time

Assigning ids and keys?

- g Use consistent hashing
- g Hashes node id (say IP address) or item id; get m-bit hash value
- g Treat hash value as int
- g Node i has item with key i if key i is part of the system

S. Venkatesan

Department of Computer Science

Fall 2007

Property of Consistent Hashing

- g For any set of N nodes and K keys, with high probability,
 - Each node is responsible for at most $(1+\epsilon)K/N$ keys
 - When a new node joins the network (or leaves), only $O(K/N)$ keys change locations
- g If implemented properly, $\epsilon = O(\log N)$
- g Can reduce ϵ to arbitrary values by each node simulating $\log N$ virtual nodes

S. Venkatesan

Department of Computer Science

Fall 2007

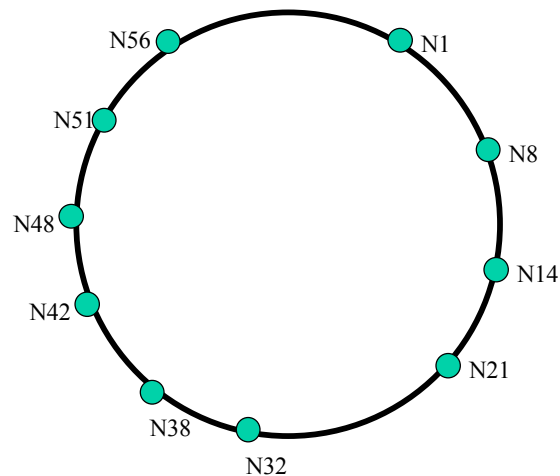
Distributed Approach

- g Organize participating nodes in a circle (modulo N)
- g Order nodes clockwise.
- g Each node knows its successor and predecessor

S. Venkatesan

Department of Computer Science

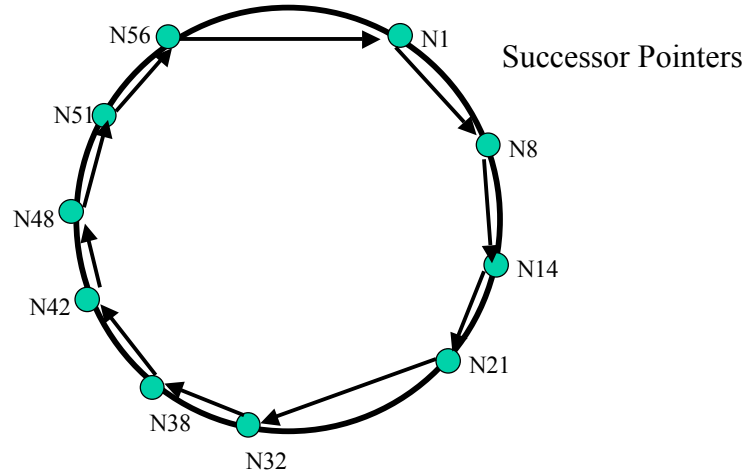
Fall 2007



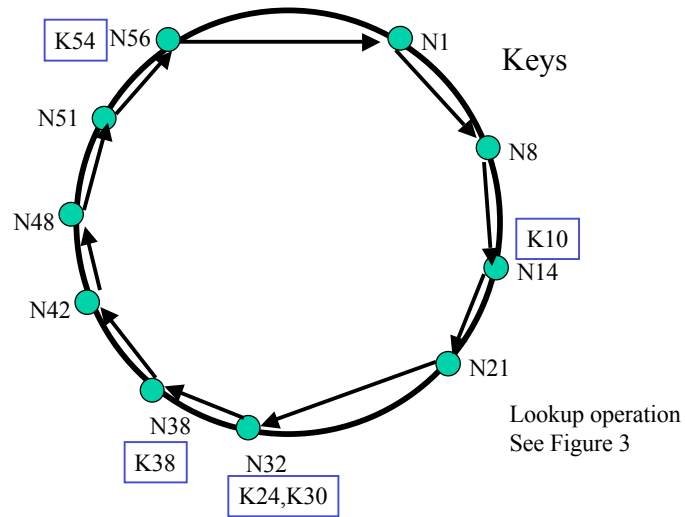
S. Venkatesan

Department of Computer Science

Fall 2007



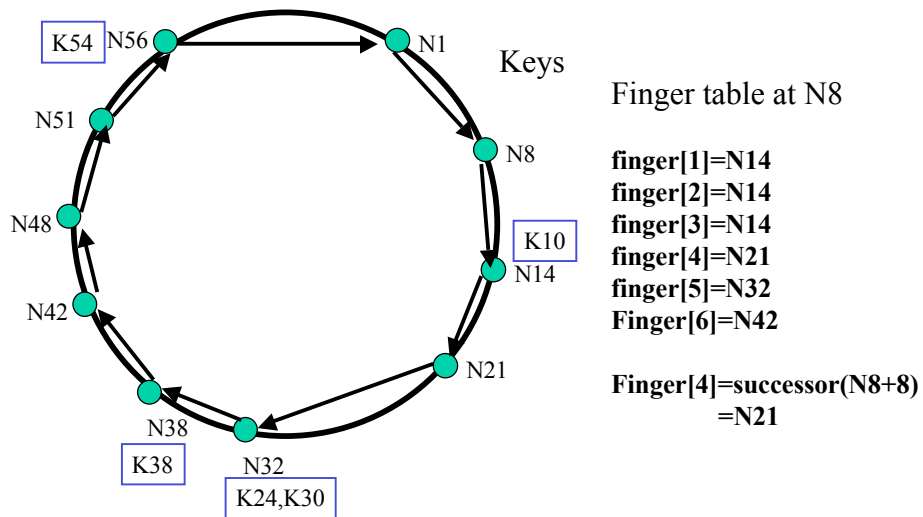
S. Venkatesan Department of Computer Science Fall 2007



S. Venkatesan Department of Computer Science Fall 2007

More efficient lookup

- g Have a finger table at each node n
- g Finger[k] = first node that succeeds $(n+2^{k-1}) \bmod N$
- g Finger[i]=successor($n+2^{i-1}$)



Finger table

- g Each node knows about small number of other nodes
- g Knows more about nodes closely following it than nodes far away
- g Finger table: not enough info to find successor[k] for arbitrary key k

Find successor operation

- g See Fig 5

Dynamic operations and failures

^g See Fig 6