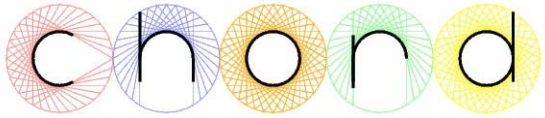


Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica, Robert Morris, David Karger,
M. Frans Kaashoek, Hari Balakrishnan

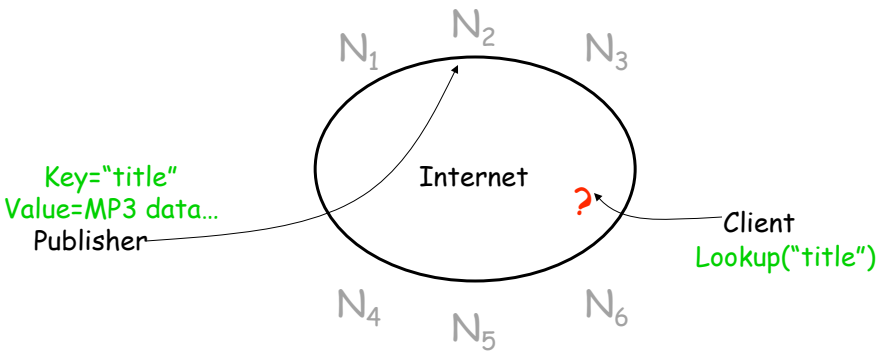
(by B. Han, with mods by A. Sussman)

(based on Robert Morris's talk at SIGCOMM 2001)

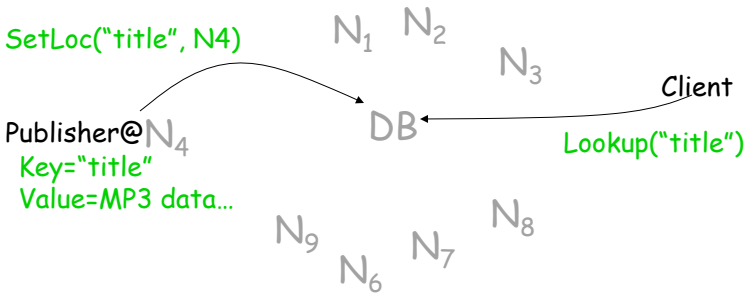
Introduction

- Core operation in peer-to-peer systems is to efficiently locate the node that stores a particular data item.
- Chord is a scalable distributed protocol for lookup in a dynamic peer-to-peer system with frequent node arrivals and departures.
- Only one operation: given a key, it maps the key onto a node.
- Key attributes:
  - Simplicity
  - provable correctness
  - provable performance.

The lookup problem

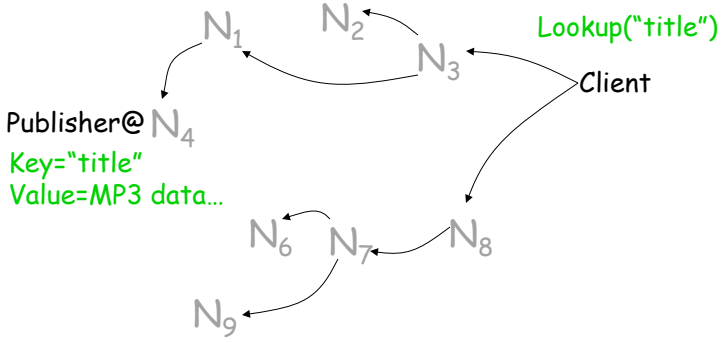


Centralized lookup (Napster)



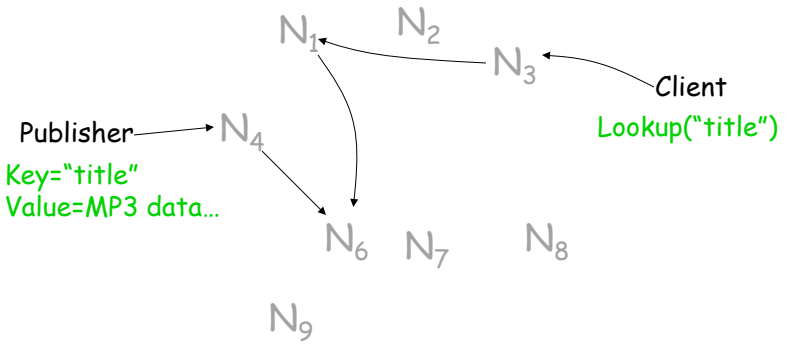
Simple, but O(N) states and a single point of failure

Flooded queries (Gnutella)



Robust, but worst case O(N) messages per lookup

Routed queries (Freenet, Chord, etc.)



Related Work

- Freenet(Clarke, Sandberg, Wiley, Hong)
- CAN (Ratnasamy, Francis, Handley, Karp, Shenker)
- Pastry (Rowstron, Druschel)
- Tapestry (Zhao, Kubiatowicz, Joseph)

… …

- Chord emphasizes simplicity

Design Objectives

- Load Balance: Consistent hash function spreads keys evenly over the nodes (Consistent hashing).
- Decentralization: Fully distributed (Robustness).
- Scalability: Lookup grows as log of number of nodes N.
- Availability: Automatically adjusts internal tables to reflect nodes joining, leaving, failing.
- Flexible Naming: Flat key space.

Application Perspective

- Chord library provides lookup(key) algorithm that yields the IP address of the node responsible for the key.
- Library notifies the application of changes in the set of keys that the node is responsible for (because of node join/leave).
- Example applications:
  - Cooperative Mirroring
  - Time-shared storage
  - Distributed indexes
  - Large-Scale combinatorial search

Routing challenges

- Define a useful key nearness metric.
- Keep the hop count small.
- Keep the routing tables small.
- Be robust to failures despite rapid changes in membership.

Chord properties

- Efficient: $O(\log(N))$ messages per lookup.
- Scalable: $O(\log(N))$ state per node.
- Robust: survives massive failures, join or leave. $O(\log^2(N))$ messages.
- An N^{th} node joins (or leaves), only $O(1/N)$ keys are moved to a different node.

Proofs are in paper / tech report.

(Assuming no malicious participants)

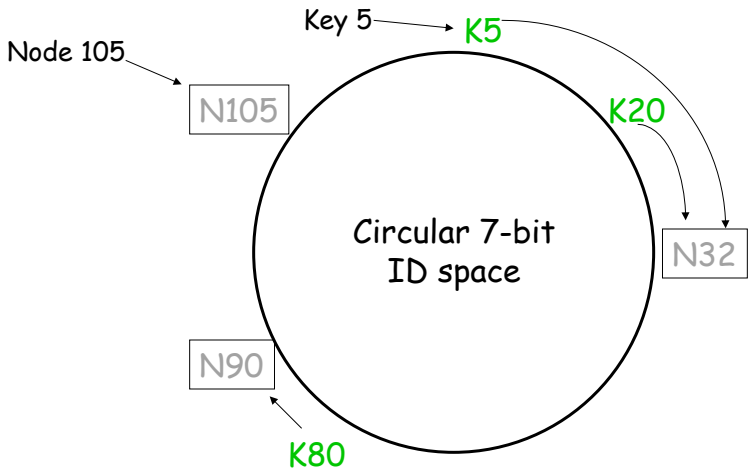
Chord overview

- Provides peer-to-peer hash lookup:
  - Lookup(key) → IP address.
  - Chord does not store the data.
  - a *structured* P2P system
- How does Chord route lookups?
- How does Chord maintain routing tables?
- How does Chord cope with changes in membership?

Chord IDs

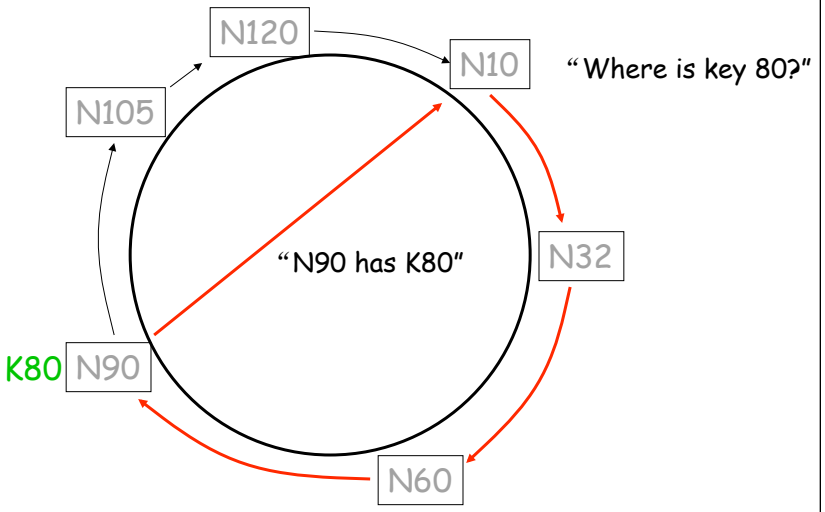
- m-bit identifier space for both keys and nodes.
- Key identifier = SHA-1(key).
  - SHA-1 is 160 bits
- Node identifier = SHA-1(IP address).
- Both are uniformly distributed in identifier space.
- How to map key IDs to node IDs?

Consistent hashing [Karger 97]



A key is stored at its successor: node with next higher ID

Basic lookup

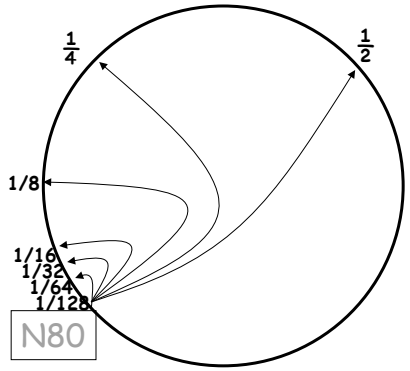


Simple lookup algorithm

```
Lookup(my-id, key-id)
  n = my successor
  if my-id < n < key-id
      call Lookup(id) on node n   // next hop
  else
      return my successor         // done
```

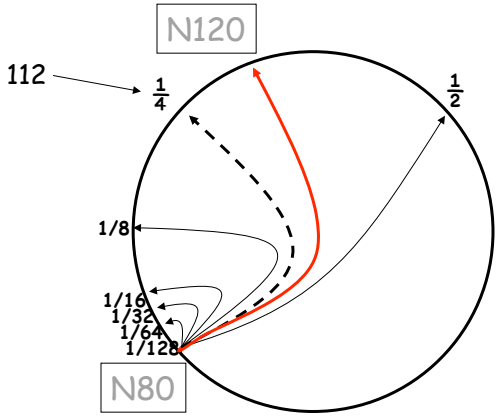
Correctness depends only on successors

"Finger table" allows O(log(N)) lookups



Every node knows *m* other nodes in the ring –
m is number bits in key

Finger *i* points to successor of $n+2^{i-1}$

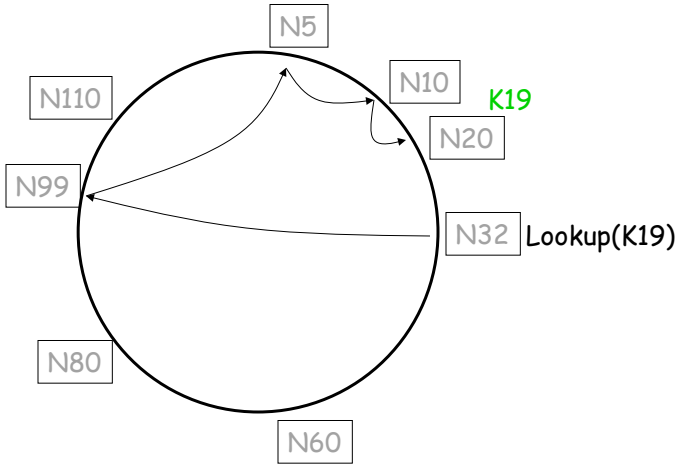


Each node knows more about portion of circle closer to it

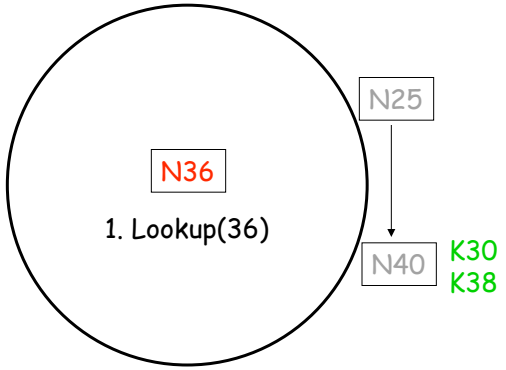
Lookup with fingers

```
Lookup(my-id, key-id)
  look in local finger table for
      highest node n s.t. my-id < n < key-id
  if n exists
      call Lookup(id) on node n   // next hop
  else
      return my successor              // done
```

Lookups take O(log(N)) hops

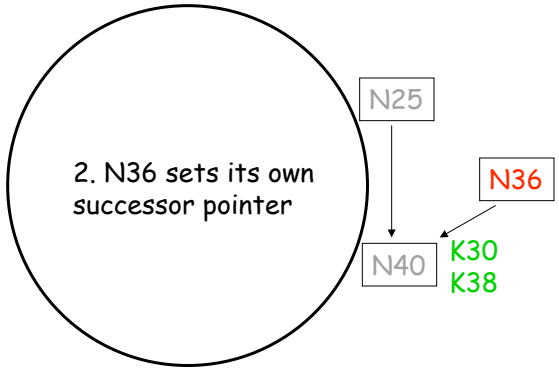


Joining: linked list insert



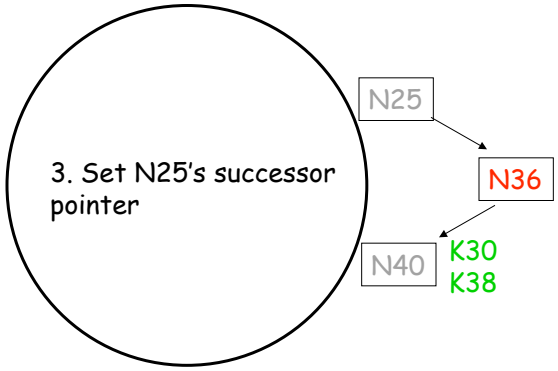
1. Each node's successor is correctly maintained.
2. For every key *k*, node *successor*(*k*) is responsible for *k*.

Join (2)



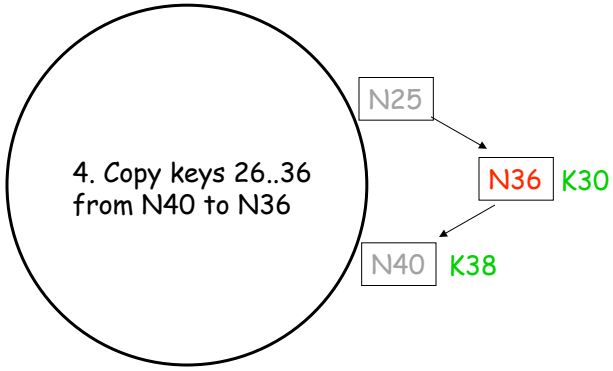
Initialize the new node finger table

Join (3)



Update finger pointers of existing nodes

Join (4)

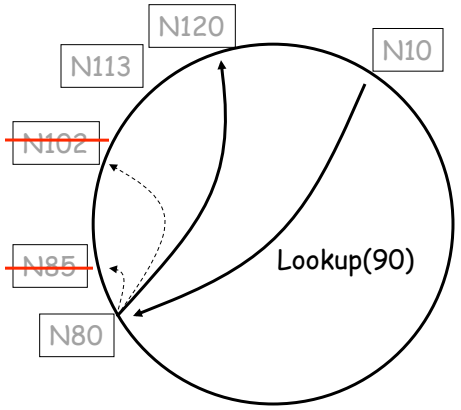


Transferring keys

Stabilization Protocol

- To handle concurrent node joins/fails/leaves.
- Keep successor pointers up to date, then verify and correct finger table entries.
- Incorrect finger pointers may only increase latency, but incorrect successor pointers may cause lookup failure.
- Nodes periodically run stabilization protocol.
- Won't correct a Chord system that has split into multiple disjoint cycles, or a single cycle that loops multiple times around the identifier space.

Failures might cause incorrect lookup



N80 doesn't know correct successor, so incorrect lookup

Solution: successor lists

- Each node knows r immediate successors.
- After failure, will know first live successor.
- Correct successors guarantee correct lookups.
- Guarantee is with some probability.
  - Can choose r to make probability of lookup failure arbitrarily small.

Choosing the successor list length

- Assume 1/2 of nodes fail.
- P(successor list all dead) = $(1/2)^r$
  - I.e. P(this node breaks the Chord ring)
  - Depends on independent failure
- P(no broken nodes) = $(1-(1/2)^r)^N$
  - $r = 2\log(N)$ makes prob. = $1 - 1/N$

Lookup with fault tolerance

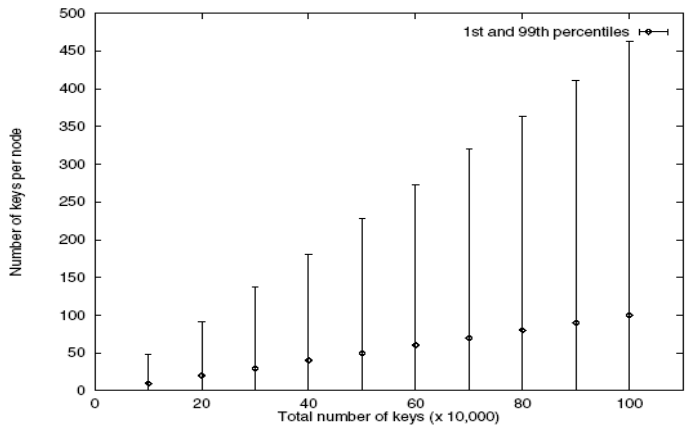
```
Lookup(my-id, key-id)
    look in local finger table and successor-list
        for highest node n s.t. my-id < n < key-id
    if n exists
        call Lookup(id) on node n          // next hop
        if call failed,
            remove n from finger table
            return Lookup(my-id, key-id)
    else return my successor               // done
```

Simulation experiments overview

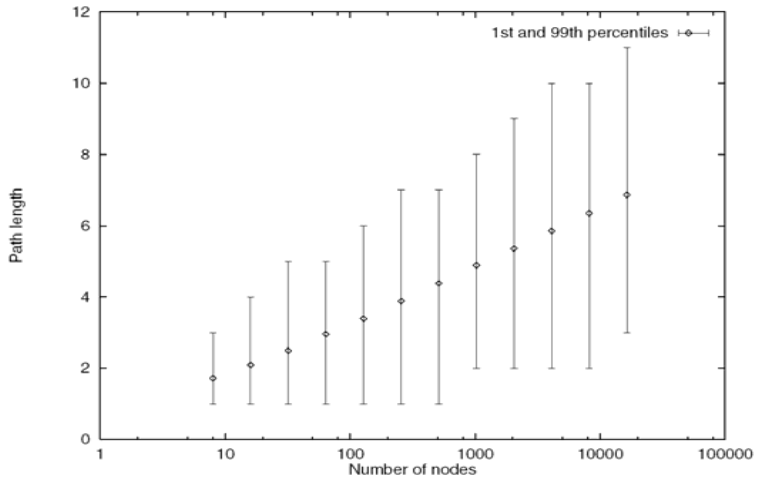
- Quick lookup in large systems.
- Low variation in lookup costs.
- Robust despite massive failure.
- Iterative implementation.
- 10,000 nodes, No. of keys range from 10^5 to 10^6.

Experiments confirm theoretical results

No. of Keys per Node

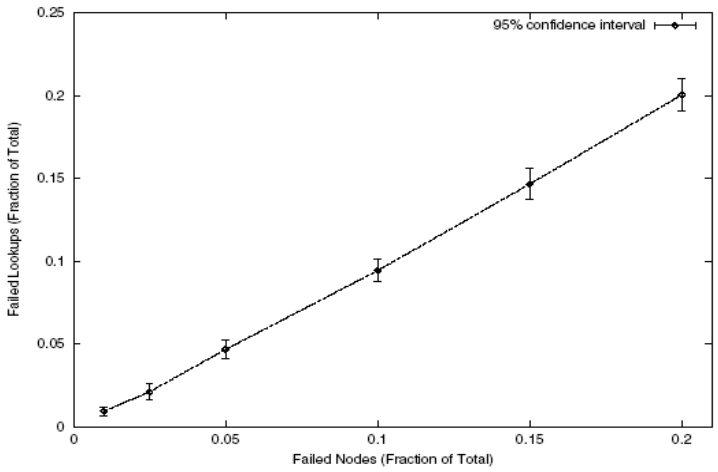


Chord lookup cost is O(log N)

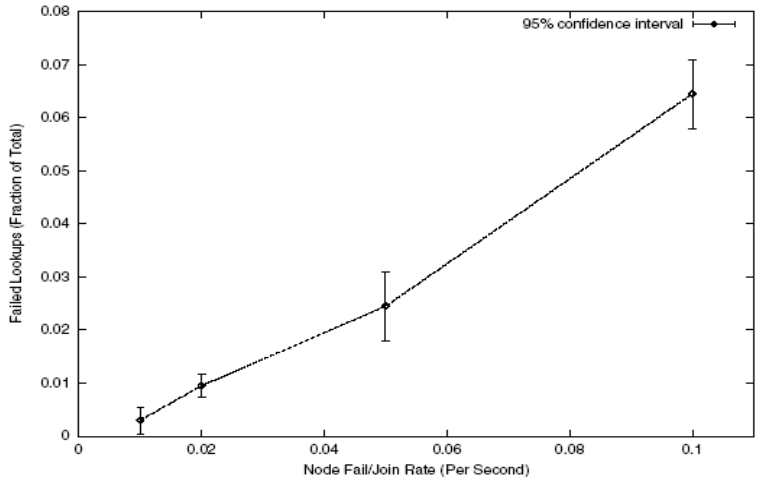


Constant is 1/2

Failed Lookups/Failed Nodes



Failed Lookups as function of Fail/Join Rate



Chord Summary

- Chord provides peer-to-peer hash lookup.
- Efficient: $O(\log(n))$ messages per lookup.
- Robust as nodes fail and join.
- Good primitive for peer-to-peer systems.

http://www.pdos.lcs.mit.edu/chord

Misc.

- Sound theoretical work (about 1173 citations as of 2006).
- Has been used in: CFS (SOSP 2001) and Ivy (OSDI 2002) file systems
- Ring Partitions might pose a problem.
- Scalability of Stabilization protocol.
  - How often does the stabilization procedure need to run?
  - How to balance consistency and network overhead?
- Virtualized ID space lacks locality characteristics.
  - Physical topology of the underlying IP network.