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

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

MIT and Berkeley
A peer-to-peer storage problem

- 1000 scattered music enthusiasts
- Willing to store and serve replicas
- How do you find the data?
The lookup problem

Key="title"
Value=MP3 data...
Publisher

N_1 N_2 N_3 N_4 N_5 N_6

Internet

Client
Lookup("title")
Centralized lookup (Napster)

Simple, but $O(N)$ state and a single point of failure
Flooded queries (Gnutella)

Robust, but worst case $O(N)$ messages per lookup
Routed queries (Freenet, Chord, etc.)

Client

Lookup("title")

Publisher

Key="title"
Value=MP3 data…
Routing challenges

- Define a useful key nearness metric
- Keep the hop count small
- Keep the tables small
- Stay robust despite rapid change

- Freenet: emphasizes anonymity
- Chord: emphasizes efficiency and simplicity
Chord properties

• Efficient: $O(\log(N))$ messages per lookup
  • $N$ is the total number of servers
• Scalable: $O(\log(N))$ state per node
• Robust: survives massive failures

• 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
• How does Chord route lookups?
• How does Chord maintain routing tables?
Chord IDs

- Key identifier = SHA-1(key)
- Node identifier = SHA-1(IP address)
- Both are uniformly distributed
- Both exist in the same ID 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

"N90 has K80"

"Where is key 80?"
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 log(N)-time lookups
Finger $i$ points to successor of $n+2^i$
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. Lookup(36)
Join (2)

2. N36 sets its own successor pointer
Join (3)

3. Copy keys 26..36 from N40 to N36
Join (4)

4. Set N25’s successor pointer

Update finger pointers in the background
Correct successors produce correct lookups
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
Choosing the successor list length

- Assume 1/2 of nodes fail
- \( P(\text{successor list all dead}) = (1/2)^r \)
  - i.e. \( P(\text{this node breaks the Chord ring}) \)
  - Depends on independent failure
- \( P(\text{no broken nodes}) = (1 - (1/2)^r)^N \)
  - \( r = 2\log(N) \) 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 \*
Chord status

- Working implementation as part of CFS
- Chord library: 3,000 lines of C++
- Deployed in small Internet testbed
- Includes:
  - Correct concurrent join/fail
  - Proximity-based routing for low delay
  - Load control for heterogeneous nodes
  - Resistance to spoofed node IDs
Experimental overview

• Quick lookup in large systems
• Low variation in lookup costs
• Robust despite massive failure
• See paper for more results

Experiments confirm theoretical results
Chord lookup cost is $O(\log N)$

Constant is $1/2$
Failure experimental setup

• Start 1,000 CFS/Chord servers
  • Successor list has 20 entries
• Wait until they stabilize
• Insert 1,000 key/value pairs
  • Five replicas of each
• Stop X% of the servers
• Immediately perform 1,000 lookups
Massive failures have little impact

\( (1/2)^6 \text{ is } 1.6\% \)
Related Work

- CAN (Ratnasamy, Francis, Handley, Karp, Shenker)
- Pastry (Rowstron, Druschel)
- Tapestry (Zhao, Kubiatowicz, Joseph)

- Chord emphasizes simplicity
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
Join: lazy finger update is OK

N2 finger should now point to N36, not N40
Lookup(K30) visits only nodes < 30, will undershoot
CFS: a peer-to-peer storage system

- Inspired by Napster, Gnutella, Freenet
- Separates publishing from serving
- Uses spare disk space, net capacity
- Avoids centralized mechanisms

- Delete this slide?
- Mention “distributed hash lookup”
CFS architecture

move later?

Block storage
Availability / replication
Authentication
Caching
Consistency
Server selection
Keyword search

Dhash distributed block store

Lookup

Chord

• Powerful lookup simplifies other mechanisms