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

Xiaozhou Li
COS 461: Computer Networks (precept 04/06/12)
Princeton University

Background

- We studied P2P file sharing in class
  - Napster
  - Gnutella
  - KaZaA
  - BitTorrent
- Today, let's learn more!
  - Chord: a scalable P2P lookup protocol
  - CFS: a distributed file system built on top of Chord
    - http://pdos.csail.mit.edu/chord

Review: distributed hash table

DHT provides the information look up service for P2P applications.
- Nodes uniformly distributed across key space
- Nodes form an overlay network
- Nodes maintain list of neighbors in routing table
- Decoupled from physical network topology

The lookup problem

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 (Chord)

Routing challenges

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

- **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) → return 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?
  - The heart of Chord protocol is "consistent hashing"

Review: consistent hashing

for data partitioning and replication

A key is stored at its successor: node with next higher ID
Identifier to node mapping example

- Node 8 maps [5, 8]
- Node 15 maps [9, 15]
- Node 20 maps [16, 20]
- ...
- Node 4 maps [59, 4]
- Each node maintains a pointer to its successor

Lookup

- Each node maintains its successor
- Route packet (ID, data) to the node responsible for ID using successor pointers

Join Operation

- Node with id=50 joins the ring via node 15
- Node 50: send join(50) to node 15
- Node 44: returns node 58
- Node 50 updates its successor to 58

Periodic Stabilize

- Node 50: periodic stabilize
- Sends stabilize message to 58
- Node 50: send notify message to 58
- Update pred = 44

Periodic Stabilize

- Node 44: periodic stabilize
- Asks 58 for pred (50)
- Node 44 updates its successor to 50
- Node 44 has a new successor (50)
- Node 44 sends a notify message to node 50
Periodic Stabilize Converges!

This completes the joining operation!

Achieving Efficiency: finger tables

Finger Table at 80

Say \( m = 7 \)

\[
\text{ith entry at peer with id } n \text{ is first peer with id } \geq n + 2^i \mod 2^m
\]

- Each node only stores \( O(\log N) \) entries
- Each lookup takes at most \( O(\log N) \) hops

Achieving Robustness

- What if nodes FAIL?
- Ring robustness: each node maintains the \( k \) (>1) immediate successors instead of only one successor
  - If smallest successor does not respond, substitute the second entry in its successor list
  - Unlikely all successors fail simultaneously
- Modifications to stabilize protocol (see paper!)

Example of Chord-based storage system

Cooperative File System (CFS)

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

DHash distributed block store

Lookup

Chord

- Powerful lookup simplifies other mechanisms

Cooperative File System (cont.)

- Block storage
  - Split each file into blocks and distribute those blocks over many servers
  - Balance the load of serving popular files
- Data replication
  - Replicate each block on \( k \) servers
  - Increase availability
  - Reduce latency (fetch from the server with least latency)
Cooperative File System (cont.)

- **Caching**
  - Caches blocks along the lookup path
  - Avoid overloading servers that hold popular data

- **Load balance**
  - Different servers may have different capacities
  - A real server may act as multiple *virtual servers*, by being hashed to several different IDs.

**Q & A**