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

Kaitlyn DeValk
Background

- What is P2P?
  - Opposite of a client-server model

- Why are they used?
  - Remember Limewire?
  - BitTorrent

- Downsides to a P2P system
  - Security, Routing/DoS attacks
What does Chord try to achieve?

- Load balancing
- Decentralization
- Scalability
- Availability
- Flexible key structure
What is Chord?

- Protocol for mapping keys to nodes
  - Efficient data lookup
- System accomplishments
  - Routing, storage, and lookups in $O(\log N)$ nodes/messages
  - Maintains data lookups in dynamic environment: $\text{Lookup}(key) = \text{IP}$
- Written in Java

*Side note - this paper was written in 2001*
Chord Use Cases

- Meant for P2P applications
- Large-scale combinatorial search
- Distributed Indexes
- Cooperative Mirroring
- Time-shared storage
Architecture

- Node & key identifiers
- Consistent hashing
  - Advantage over traditional hash tables in remapping keys
  - Orders identifiers in identifier circle mod $2^m$
  - Each node is responsible for $(1+\varepsilon)K/N$ keys
- Scalable key location
  - Finger table: $s = (n+2^{i-1}) \mod 2^m$
    - $m$ most entries, $O(\log N)$-time lookups
  - Allows nodes to store as little information as necessary (successor pointers)
Architecture (Cont’d)

- Node joins (predecessor pointers)
  - Initialize predecessor
  - Update finger tables & predecessors: takes $O(\log^2 N)$ messages
  - Notify actual software so state can be transferred to new node

- For any joins/leaves on a given node
  - Joins - certain keys assigned to successor get assigned to new node
  - Leaves - all keys assigned to successor
  - Only $O(1/N)$ keys are moved
Finger Tables & Lookups

- Stores info about small number of other nodes
  - Knows more about nodes closer to it on the “circle”
- Not enough info to find successor of arbitrary key
- What happens if a given key isn’t in a node’s finger table?
  - At most $O(\log N)$ nodes must be contacted
Basic Lookups & Successors

- Three nodes, three keys

- Lookup(2) = 3

- Successor(2) = 3
  - \((2 + 2^{(1-1)}) \mod 2^2\)
  - Assume 3-bit ID
  - First successor of node 2
Concurrent Jobs & Failures

- Node’s $r$ successors fail with $1/N$ probability
  - Successor list of length $r = \Theta(\log N)$
- Lookup fails are handled with “stabilization” protocol
  - Used to ensure successor pointers are updated
  - Also how new nodes are “noticed” by the network
  - Won’t correct disjoint systems or ones that infinitely loop the identifier circle
- Nodes maintain nearest successor’s list in event of node failure
  - Even in failed networks $\Theta(\log N)$ to find successors
Results

- Load balancing isn’t perfect w/Chord
- Simultaneous node failures didn’t significantly affect lookups
- Stabilization protocol seems to work (mostly)
- Latency didn’t grow exponentially w/node increase -> Stability!
Where did Chord come up short?

- Not fast enough for some applications
- Routing tables/hop count
- Simultaneous nodes joining/leaving
- Occasionally linear lookups/scans can occur
- Overall security not really mentioned or touched by Chord protocol
Overall Opinion on the Paper

- Lots of math, made it hard to follow at times
- Not sure what was meant by the “external mechanism” handling new nodes learning about existing node’s identities
- Didn’t really go in-depth on differences between leave/join functionality even though they were implied
- Found a UMD grad project from 2003 that improved & implemented the Chord algorithms
Questions?