LOCATING DATA ON THE NETWORK: P2P NETWORKS, CHORD, AND DYNAMODB

Feb 22, 2022
George Porter
ATTRIBUTION

• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license

• These slides incorporate material from:
  • Christo Wilson, NEU (used with permission)
  • Kyle Jamieson, Princeton
  • Tanenbaum and Van Steen, 3rd edition
ANNOUNCEMENTS

Reading for Thursday:
“The Tail at Scale” (Dean and Barroso), linked off canvas

Optional reading for today:
DynamoDB paper linked off canvas

Today:
Finish up Chord, talk a bit about DynamoDB
Figure 5-4. Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.
AN ASIDE: IS LOG(N) FAST OR SLOW?

- For a million nodes, it’s 20 hops

- If each hop takes 50 milliseconds, lookups take a second

- If each hop has 10% chance of failure, it’s a couple of timeouts

- So in practice log(n) is better than O(n) but not great
JOINING: LINKED LIST INSERT

1. Lookup(36)
2. N36 sets its own successor pointer
JOIN (3)

3. Copy keys 26..36 from N40 to N36
NOTIFY MESSAGES MAINTAIN PREDECESSORS
"My predecessor is N36."
JOINING: SUMMARY

- Predecessor pointer allows link to new node
- Update finger pointers in the background
- Correct successors produce correct lookups
WHAT CHORD GOT RIGHT

- **Consistent hashing**
  - Elegant way to divide a workload across machines
  - Very useful in clusters: actively used today in Amazon Dynamo and other systems

- **Replication** for high availability, efficient recovery after node failure

- **Incremental scalability**: “add nodes, capacity increases”

- **Self-management**: minimal configuration

- **Unique trait**: no single server to shut down/monitor