CSCI 339
Distributed Systems

Lecture 22
DHTs and Chord
May 1, 2023
Administrative Details

• Checkpoint (~1 page) due TOMORROW on Glow
  • I read the proposals. If you didn’t hear from me, I approve!
  • Checkpoint: What have you done so far? What do you have left? Are you having trouble with anything? Has the scope of your project changed in any way? **What will you evaluate?**
  • Please come talk to me/Jae if you are stuck or need guidance!
• Bitcoin writeup is NOT REQUIRED
  • Skim the paper and/or watch the optional video
  • No writeup required
• Graded Bookstore and Hadoop coming this week!
Next Week

• Final project presentations
  • 21 groups
  • Class next week: 12:30-2:25 (w/pizza on Thur)?
  • Snack ideas for Mon??
  • 8-9 min per group (enforced)!
  • More details on Thursday
  • Please respond to Google form about availability!
  • Vote for favorite project, nominate winner for Ward prize

• Invited to stick around and see Graphics demos at 2:35
Final Project Groups

1. Ashton and Dylan
2. Emma C, Ben W, Gregor
3. Roxanne
4. Meredith and Emily
5. Ben Shapiro and Benjamin
6. Ruby and Levi
7. Whit
8. Sabiha
9. Brian and Victor
10. Yousef
11. Tai, Enoch, and Gerardo
12. Petros
13. Sophie, Jules, Emma N
14. Bri and Zach
15. Nathan
16. Mark and David
17. Atlas
18. Will
19. Jason
20. Daniel
21. Ye

Any changes???
Last Class

• Peer-to-peer (P2P) networking overview
  • Background
  • Case studies:
    • Napster
    • Gnutella
    • Kazaa
    • BitTorrent
  • Discussion: it’s all about the tradeoffs!
• There are ~three key distributed computing paradigms:
  • Client-server architectures
    • You provide a service (in the form of a server) that other people can use
      – Web server, Multi-tier bookstore
  • Cluster/Cloud computing
    • More “distributed” approach to parallel processing
    • Lots of computers (usually in a cluster/LAN/cloud) working together towards a common goal
      – MapReduce/Hadoop
  • P2P computing
    • No notion of client/server; all machines are equal
    • Wide-area network issues
      – Final project?
Today’s Outline

• Learn about distributed hash tables (DHTs)
  • How do you find a given data item in a P2P system?
  • This “lookup problem” is tricky
  • DHTs offer one possible solution

• There are many DHTs (very popular in 2001-2005)
  • CAN, Tapestry, Pastry, Chord, Bamboo
  • We’re going to focus on Chord (the “first” one)
Wide-Area Computing Challenges

• Wide-area network conditions
  • Links have lower speeds: more latency and less (or at least variable) bandwidth
  • Links are more lossy and failure-prone (and there are more of them to traverse between two hosts)
  • Unpredictable network congestion and bottlenecks
  • Many routers are involved (introduce delay)
  • Network links are shared

• Less control over machines, more heterogeneity
  • Hosts are located in different cities, countries, continents
  • Hosts are run by different schools, labs, people

• Less control over computing environment
Wide-Area Computing Benefits

• Lots of computational resources out there
  • Why limit ourselves to just machines we physically control?
  • If we can build a system that can tolerate wide-area variability, we have many more options

• Shared platforms
  • Our labs have ~40 computers
  • UMass has 300+ computers
  • High probability that UMass is not using all 300 all the time
  • By sharing resources, both of us benefit (ok, so maybe we benefit a bit more…)
  • Increase overall utility of machines and decrease cost
Motivation for DHTs

• Let’s think back to Napster, Gnutella, and BitTorrent
• Searching in P2P file sharing systems across the wide area is a challenge (no common file system)
  • Napster solved it using a central index, which was also a single point of failure
  • Gnutella used a fully decentralized (slow) process that used lots of bandwidth
  • BitTorrent didn’t even attempt to solve it!
Motivation for DHTs

How to find data in a distributed file sharing system?

“Lookup” is the main problem!

Publisher
Key=“LetItBe”
Value=MP3 data

Client
Lookup(“LetItBe”)

Slide content based on material from Daniel Figueiredo and Robert Morris
Centralized Solution

Central server (Napster)

Publisher
Key="LetItBe"
Value=MP3 data

Client
Lookup("LetItBe")
Distributed Solution (1)

Flooding (Gnutella, Morpheus, etc.)

Publisher
Key="LetItBe"
Value=MP3 data

Client
Lookup("LetItBe")
Distributed Solution (2)

Routed messages (Freenet, Tapestry, Chord, CAN, etc.)

Publisher
Key=“LetItBe”
Value=MP3 data

Client
Lookup(“LetItBe”)
Routing Challenges

• What makes routing “lookup messages” hard
  • Define a useful “key nearness” metric
  • Keep the hop count small
  • Keep the routing tables “right size”
  • Stay robust despite rapid changes in membership

• Chord: emphasizes efficiency and simplicity

• Note: Chord is not a P2P file sharing system!
Chord Overview

• Provides peer-to-peer hash lookup service (basically a distributed index):
  • Lookup(key) → IP address
  • It’s just a hash table!
  • Note: Chord does not store the data being looked up!

• How does Chord locate a node?
• How does Chord maintain routing tables?
• How does Chord cope with changes in membership?
Chord Properties

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

- Formal proofs are in the original 2001 paper
  - Assume no malicious participants
Chord IDs

- \( m \) bit identifier space for both keys and nodes
- Key identifier = SHA-1(key)

\[
\text{Key=“LetItBe”} \xrightarrow{\text{SHA-1}} \text{ID=60}
\]

- Node identifier = SHA-1(IP address)

\[
\text{IP=“137.165.10.100”} \xrightarrow{\text{SHA-1}} \text{ID=123}
\]

- Both are uniformly distributed and exist in same ID space
- Goal: How to map key IDs to node IDs?
Consistent Hashing [Karger97]

- Given a set of $n$ nodes, a consistent hash function will map keys (e.g., filenames) uniformly across the nodes
  - Load balancing!

- Nice feature of consistent hashing for node addition:
  - Only $1/n$ keys must be reassigned to new nodes who join
Consistent Hashing

Circular m-bit ID space
Consistent Hashing

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

Circular m-bit ID space

IP="137.165.10.100"

Key="LetItBe"
Consistent Hashing

Hash("LetItBe") = K60

Where is “LetItBe”? Hash("LetItBe") = K60

“N90 has K60”

Advantages? Disadvantages?
Chord: Basic Lookup

- Every node knows only its successor in the ring

Where is “LetItBe”?  
Hash(“LetItBe”) = K60

“N90 has K60”
“Finger Tables”

- Every node knows up to $m$ other nodes in the ring
- Increase distance exponentially
- $m=7$ in this example
“Finger Tables”

- Finger $i$ points to successor of $n+2^{i-1}$
- $i^{th}$ entry in n’s finger table has ID $> (n+2^{i-1}) \mod 2^m$
Lookups are Faster

N110 N10 N20 N32

N99 N80 N5

Lookup(K19)
Lookups are Faster
Joining the Ring

- Three step process:
  - Initialize all fingers of new node
  - Update fingers of existing nodes
  - Transfer keys from successor to new node

- Less aggressive mechanism (lazy finger update):
  - Initialize only the finger to successor node
  - Periodically verify immediate successor, predecessor
  - Periodically refresh finger table entries
Joining the Ring - Step 1

- Initialize the new node’s (N36) finger table
  - Locate any node $p$ in the ring
  - Ask node $p$ to lookup fingers of new node N36
  - Return results to new node

1. Lookup(37,38,40,....,100,164)
Joining the Ring - Step 2

- Updating fingers of existing nodes
  - New node calls update function on existing nodes
  - Existing nodes can recursively update fingers of other nodes
  - N36 sets successor pointer to be N40
  - N20 sets successor pointer to be N36
Joining the Ring - Step 3

• Transfer keys from successor node to new node
  
  • Only keys in the range are transferred

Copy keys 21…36 from N40 to N36

• Note: When a node leaves ring, all keys are copied to successor
Handing Failures

• Failure of nodes might cause incorrect lookup

• N80 doesn’t know correct successor, so lookup fails

• What should we do?
Handing Failures

• Use successor list (in addition to finger table)
  • 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
Evaluation Overview

• Quick lookup in large systems
• Low variation in lookup costs
• Robust despite massive failure

• Experiments confirm theoretical results (which is always a good thing)
Cost of lookup

- Cost is $O(\log N)$ as predicted by theory
  - Constant is $1/2$
Robustness

- Start with 1000 peers
- Insert 1000 key/value pairs (and replicate each 5 times)
- Stop X% of peers
- Perform 1000 lookups

Massive failures have little impact!
Effectiveness of Load Balancing

![Graph showing the relationship between total number of keys and number of keys per node, with 1st and 99th percentiles indicated.]

- X-axis: Total number of keys (x 10,000)
- Y-axis: Number of keys per node

The graph illustrates the distribution of keys per node across different totals of keys, highlighting the 1st and 99th percentiles.
Path Length of Lookup

![Graph showing the path length of lookup against the number of nodes. The graph includes a trend line and error bars representing the 1st and 99th percentiles. The x-axis represents the number of nodes, ranging from 1 to 100,000, and the y-axis represents the path length, ranging from 0 to 12.]
Distribution of Path Length
(4096 nodes)
Discussion

• Limitations? Problems? Questions?
• Locality with respect to the underlying network?
  • From Mass, first lookup goes to Australia, second to Europe, third to Asia
• Even $O(\log n)$ steps too many for routing in large networks?
• Single popular key mapping to a single node?
• What about search?
• How does replication fit into the picture?