15-744: Computer Networking

P2P/DHT
Overview

• P2P Lookup Overview

• Centralized/Flooded Lookups

• Routed Lookups – Chord

• Comparison of DHTs
Peer-to-Peer Networks

• Typically each member stores/provides access to content
• Basically a replication system for files
  • Always a tradeoff between possible location of files and searching difficulty
  • Peer-to-peer allow files to be anywhere → searching is the challenge
  • Dynamic member list makes it more difficult
• What other systems have similar goals?
  • Routing, DNS
The Lookup Problem

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

Internet

N_1 \rightarrow N_2 \rightarrow N_3

N_4 \rightarrow N_5 \rightarrow N_6

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 (Chord, etc.)

Publisher

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

N1 → N2 → N3

Client

Lookup("title")

N4

N6 → N7

N8

N9
Overview

• P2P Lookup Overview

• Centralized/Flooded Lookups

• Routed Lookups – Chord

• Comparison of DHTs
Centralized: Napster

• Simple centralized scheme → motivated by ability to sell/control

• How to find a file:
  • On startup, client contacts central server and reports list of files
  • Query the index system → return a machine that stores the required file
    • Ideally this is the closest/least-loaded machine
  • Fetch the file directly from peer
Centralized: Napster

• Advantages:
  • Simple
  • Easy to implement sophisticated search engines on top of the index system

• Disadvantages:
  • Robustness, scalability
  • Easy to sue!
Flooding: Old Gnutella

• On startup, client contacts any servent (server + client) in network
  • Servent interconnection used to forward control (queries, hits, etc)
• Idea: broadcast the request
• How to find a file:
  • Send request to all neighbors
  • Neighbors recursively forward the request
  • Eventually a machine that has the file receives the request, and it sends back the answer
• Transfers are done with HTTP between peers
Flooding: Old Gnutella

• Advantages:
  • Totally decentralized, highly robust

• Disadvantages:
  • Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)
  • Especially hard on slow clients
    • At some point broadcast traffic on Gnutella exceeded 56kbps – what happened?
    • Modem users were effectively cut off!
Flooding: Old Gnutella Details

• Basic message header
  • Unique ID, TTL, Hops

• Message types
  • Ping – probes network for other servents
  • Pong – response to ping, contains IP addr, # of files, # of Kbytes shared
  • Query – search criteria + speed requirement of servent
  • QueryHit – successful response to Query, contains addr + port to transfer from, speed of servent, number of hits, hit results, servent ID
  • Push – request to servent ID to initiate connection, used to traverse firewalls

• Ping, Queries are flooded
• QueryHit, Pong, Push reverse path of previous message
Flooding: Old Gnutella Example

Assume: m1’s neighbors are m2 and m3; m3’s neighbors are m4 and m5;...
Flooding: Gnutella, Kazaa

- Modifies the Gnutella protocol into two-level hierarchy
  - Hybrid of Gnutella and Napster
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
  - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across supernodes
- Disadvantages
  - Kept a centralized registration → allowed for law suits 😞
Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord
- Comparison of DHTs
Routing: Structured Approaches

- Goal: make sure that an item (file) identified is always found in a reasonable # of steps
- Abstraction: a distributed hash-table (DHT) data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Proposals
  - CAN (ICIR/Berkeley)
  - Chord (MIT/Berkeley)
  - Pastry (Rice)
  - Tapestry (Berkeley)
  - …
Routing: Chord

• Associate to each node and item a unique id in an uni-dimensional space

• Properties
  • Routing table size $O(\log(N))$, where $N$ is the total number of nodes
  • Guarantees that a file is found in $O(\log(N))$ steps
Aside: Hashing

• Advantages
  • Let nodes be numbered 1..m
  • Client uses a **good** hash function to map a URL to 1..m
  • Say hash (url) = \( x \), so, client fetches content from node \( x \)
  • No duplication – not being fault tolerant.
  • One hop access
  • Any problems?
    • What happens if a node goes down?
    • What happens if a node comes back up?
    • What if different nodes have different views?
Robust hashing

• Let 90 documents, node 1..9, node 10 which was dead is alive again

• % of documents in the wrong node?
  • 10, 19-20, 28-30, 37-40, 46-50, 55-60, 64-70, 73-80, 82-90
  • *Disruption coefficient* = ½

• Unacceptable, use consistent hashing – idea behind Akamai!
Consistent Hash

• “view” = subset of all hash buckets that are visible

• Desired features
  • Balanced – in any one view, load is equal across buckets
  • Smoothness – little impact on hash bucket contents when buckets are added/removed
  • Spread – small set of hash buckets that may hold an object regardless of views
  • Load – across all views # of objects assigned to hash bucket is small
Consistent Hash – Example

- Construction
  - Assign each of $C$ hash buckets to random points on mod $2^n$ circle, where, hash key size = $n$.
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket

- Smoothness $\Rightarrow$ addition of bucket does not cause much movement between existing buckets
- Spread & Load $\Rightarrow$ small set of buckets that lie near object
- Balance $\Rightarrow$ no bucket is responsible for large number of objects
Routing: Chord Basic Lookup

“Where is key 80?”

“N90 has K80”
Routing: Finger table - Faster Lookups

N80

1/128
1/64
1/32
1/16
1/8

1/4

1/2
Routing: Chord Summary

- Assume identifier space is $0...2^m$
- Each node maintains
  - Finger table
    - Entry $i$ in the finger table of $n$ is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by $id$ is stored on the successor node of $id$
Routing: Chord Example

• Assume an identifier space 0..8
• Node n1:(1) joins all entries in its finger table are initialized to itself

<table>
<thead>
<tr>
<th>i</th>
<th>id + 2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Routing: Chord Example

- Node n2: (3) joins
Routing: Chord Example

- Nodes n3:(0), n4:(6) join
Routing: Chord Examples

- Nodes: n1(1), n2(3), n3(0), n4(6)
- Items: f1:(7), f2:(2)
Routing: Query

- Upon receiving a query for item $id$, a node
- Check whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed $id$
What can DHTs do for us?

- Distributed object lookup
  - Based on object ID
- De-centralized file systems
  - CFS, PAST, Ivy
- Application Layer Multicast
  - Scribe, Bayeux, Splitstream
- Databases
  - PIER
Overview

• P2P Lookup Overview
• Centralized/Flooded Lookups
• Routed Lookups – Chord
• Comparison of DHTs
Comparison

• Many proposals for DHTs
  • Tapestry (UCB) -- Symphony (Stanford) -- 1hop (MIT)
  • Pastry (MSR, Rice) -- Tangle (UCB) -- conChord (MIT)
  • Chord (MIT, UCB) -- SkipNet (MSR, UW) -- Apocrypha (Stanford)
  • CAN (UCB, ICSI) -- Bamboo (UCB) -- LAND (Hebrew Univ.)
  • Viceroy (Technion) -- Hieras (U.Cinn) -- ODRI (TexasA&M)
  • Kademlia (NYU) -- Sprout (Stanford)
  • Kelips (Cornell) -- Calot (Rochester)
  • Koorde (MIT) -- JXTA’s (Sun)

• What are the right design choices? Effect on performance?
Deconstructing DHTs

Two observations:

1. Common approach
   - N nodes; each labeled with a virtual identifier (128 bits)
   - define “distance” function on the identifiers
   - routing works to reduce the distance to the destination

2. DHTs differ primarily in their definition of “distance”
   - typically derived from (loose) notion of a routing geometry
DHT Routing Geometries

- Geometries:
  - Tree (Plaxton, Tapestry)
  - Ring (Chord)
  - Hypercube (CAN)
  - XOR (Kademlia)
  - Hybrid (Pastry)

- What is the impact of geometry on routing?
Geometry

- nodes are leaves in a binary tree
- distance = height of the smallest common subtree
- logN neighbors in subtrees at distance 1, 2, ..., logN
Hypercube (CAN)

Geometry

- nodes are the corners of a hypercube
- **distance** = #matching bits in the IDs of two nodes
- \( \log N \) neighbors per node; each at distance=1 away
Ring (Chord)

Geometry

- nodes are points on a ring
- **distance** = numeric distance between two node IDs
- $\log N$ neighbors exponentially spaced over $0 \ldots N$
Hybrid (Pastry)

Geometry:

- combination of a tree and ring
- two distance metrics
- default routing uses tree; fallback to ring under failures
  - neighbors picked as on the tree
XOR (Kademlia)

Geometry:

- `distance(A,B) = A XOR B`
- `logN` neighbors per node spaced exponentially
- not a ring because there is no single consistent ordering of all the nodes
Geometry’s Impact on Routing

- Routing
  - Neighbor selection: how a node picks its routing entries
  - Route selection: how a node picks the next hop

- Proposed metric: flexibility
  - amount of freedom to choose neighbors and next-hop paths
    - FNS: flexibility in neighbor selection
    - FRS: flexibility in route selection
  - intuition: captures ability to “tune” DHT performance
  - single predictor metric dependent only on routing issues
• Chord algorithm picks neighbor closest to destination
• A different algorithm picks the best of alternate paths
FNS for Ring Geometry

- Chord algorithm picks $i^{th}$ neighbor at $2^i$ distance
- A different algorithm picks $i^{th}$ neighbor from $[2^i, 2^{i+1})$
## Flexibility: at a Glance

<table>
<thead>
<tr>
<th>Flexibility</th>
<th>Ordering of Geometries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbors</td>
<td>Hypercube $&lt;&lt;$ Tree, XOR, Ring, Hybrid</td>
</tr>
<tr>
<td>(FNS)</td>
<td>$(1)$</td>
</tr>
<tr>
<td></td>
<td>$2^{i-1}$</td>
</tr>
<tr>
<td>Routes</td>
<td>Tree $&lt;&lt;$ XOR, Hybrid $&lt;$ Hypercube $&lt;$ Ring</td>
</tr>
<tr>
<td>(FRS)</td>
<td>$(1)$</td>
</tr>
<tr>
<td></td>
<td>$(\log N/2)$</td>
</tr>
<tr>
<td></td>
<td>$(\log N/2)$</td>
</tr>
<tr>
<td></td>
<td>$(\log N)$</td>
</tr>
</tbody>
</table>
Validate over three performance metrics:

1. resilience
2. path latency
3. path convergence

Metrics address two typical concerns:

- ability to handle node failure
- ability to incorporate proximity into overlay routing
Analysis of Static Resilience

Two aspects of robust routing

• Dynamic Recovery: how quickly routing state is recovered after failures

• Static Resilience: how well the network routes before recovery finishes
  • captures how quickly recovery algorithms need to work
  • depends on FRS

Evaluation:

• Fail a fraction of nodes, without recovering any state
• Metric: % Paths Failed
Does flexibility affect static resilience?

Flexibility in Route Selection matters for Static Resilience
Which is more effective, FNS or FRS?

Plain \ll FRS \ll FNS \approx FNS+FRS

Neighbor Selection is much better than Route Selection
Does Geometry affect performance of FNS or FRS?

No, performance of FNS/FRS is independent of Geometry.
A Geometry’s support for neighbor selection is crucial.
Understanding DHT Routing: Conclusion

• What makes for a “good” DHT?
  • one answer: a flexible routing geometry

• Result: Ring is most flexible
  • Why not the Ring?
Next Lecture

- Data-oriented networking
- Required readings
  - Content-centric networking (read)
  - Redundancy elimination (skim)
- Optional readings
  - DOT, DONA, DTN
Overview

• Bittorrent
Peer-to-Peer Networks: BitTorrent

- BitTorrent history and motivation
  - 2002: B. Cohen debuted BitTorrent
  - Key motivation: popular content
    - Popularity exhibits temporal locality (Flash Crowds)
    - E.g., Slashdot/Digg effect, CNN Web site on 9/11, release of a new movie or game
  - Focused on efficient fetching, not searching
    - Distribute same file to many peers
    - Single publisher, many downloaders
  - Preventing free-loading
BitTorrent: Simultaneous Downloading

• Divide large file into many pieces
  • Replicate different pieces on different peers
  • A peer with a complete piece can trade with other peers
  • Peer can (hopefully) assemble the entire file

• Allows simultaneous downloading
  • Retrieving different parts of the file from different peers at the same time
  • And uploading parts of the file to peers
  • Important for very large files
BitTorrent: Tracker

- Infrastructure node
  - Keeps track of peers participating in the torrent
- Peers register with the tracker
  - Peer registers when it arrives
  - Peer periodically informs tracker it is still there
- Tracker selects peers for downloading
  - Returns a random set of peers
  - Including their IP addresses
  - So the new peer knows who to contact for data
- Can have “trackerless” system using DHT
BitTorrent: Chunks

- Large file divided into smaller pieces
  - Fixed-sized chunks
  - Typical chunk size of 256 Kbytes
- Allows simultaneous transfers
  - Downloading chunks from different neighbors
  - Uploading chunks to other neighbors
- Learning what chunks your neighbors have
  - Periodically asking them for a list
- File done when all chunks are downloaded
Self-certifying Names

- A piece of data comes with a public key and a signature.
- Client can verify the data did come from the principal by
  - Checking the public key hashes into P, and
  - Validating that the signature corresponds to the public key.
- Challenge is to resolve the flat names into a location.
BitTorrent: Overall Architecture

Web Server

Tracker

Web page with link to .torrent

Peer [Leech]

Downloader

“US”

Peer [Leech]

C

Peer [Seed]
BitTorrent: Overall Architecture

Web Server

Tracker

Get-announce

A

Peer
[Leech]
Downloader
“US”

B

Peer
[Leech]

C

Peer
[Seed]
BitTorrent: Overall Architecture

- **Web Server**: Web page with link to .torrent
- **Tracker**
- **Response-peer list**
- **Peer [Leech]**
- **Downloader**
- **“US”**
- **Peer [Leech]**
- **Peer [Seed]**
- **C**
BitTorrent: Overall Architecture

Web Server

Tracker

Web page with link to .torrent

Peer [Leech]

Downloader

“US”

Peer [Leech]

Peer [Seed]

A

B

C

Shake-hand

Shake-hand

Shake-hand
BitTorrent: Overall Architecture

Web Server

Tracker

Peer [Leech]
Downloader
“US”

Peer

Peer [Leech]

Peer [Seed]

Web page with link to .torrent

A

B

C

pieces

pieces
BitTorrent: Overall Architecture

Web Server

Tracker

Web page with link to .torrent

Peer

[Leech]

Downloader

“US”

Peer

[Seed]

Peer

[Leech]

B

C

pieces

pieces

pieces
BitTorrent: Overall Architecture

Web Server

Tracker

Web page with link to .torrent

Get-announce

Response-peer list

pieces

pieces

pieces

Peer [Leech]

Downloader "US"

Peer [Leech]

Peer [Seed]