Indranil Gupta (Indy)
September 27, 2012
Lecture 10
Peer-to-peer Systems II

Reading: Chord paper on website (Sec 1-4, 6-7)
Two types of P2P Systems

Systems that work well in practice but with no big/famous names
• *Non-academic P2P systems*
  e.g., Gnutella, Napster (previous lecture)

Systems with big/famous names from academia, with varied uses
• *Academic P2P systems*
  e.g., Chord (this lecture)
DHT=Distributed Hash Table

- A hash table allows you to insert, lookup and delete objects with keys
- A distributed hash table allows you to do the same in a distributed setting (objects=files)
- DHTs are inspiration for key-value store in a cloud
- Performance Concerns:
  - Load balancing
  - Fault-tolerance
  - Efficiency of lookups and inserts
- Napster, Gnutella, FastTrack are all DHTs (sort of)
- So is Chord, a structured peer to peer system that we study next
## Comparative Performance

<table>
<thead>
<tr>
<th></th>
<th>Memory</th>
<th>Lookup Latency</th>
<th>#Messages for a lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Napster</strong></td>
<td>$O(1)$ ((O(N))@server)</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Gnutella</strong></td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>
## Comparative Performance

<table>
<thead>
<tr>
<th></th>
<th>Memory</th>
<th>Lookup Latency</th>
<th>#Messages for a lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Napster</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td></td>
<td>($O(N)@$server)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gnutella</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Chord</td>
<td>$O(\log(N))$</td>
<td>$O(\log(N))$</td>
<td>$O(\log(N))$</td>
</tr>
</tbody>
</table>
Chord

- **Developers**: I. Stoica, D. Karger, F. Kaashoek, H. Balakrishnan, R. Morris, Berkeley and MIT
- Intelligent choice of neighbors to reduce latency and message cost of routing (lookups/inserts)
- Uses *Consistent Hashing* on node’s (peer’s) address
  - SHA-1(ip_address, port) → 160 bit string
  - Truncated to $m$ bits
  - Called peer *id* (number between 0 and $2^m - 1$)
  - Not unique but id conflicts very unlikely ($m \sim 128$)
  - Can then map peers to one of $2^m$ logical points on a circle
Ring of peers

Say $m=7$

6 nodes
Peer pointers (1): *successors*

Say $m=7$

N112 \rightarrow 0 \rightarrow N16

N96 \rightarrow 0 \rightarrow N32

N80 \rightarrow 0 \rightarrow N45

(similarly predecessors)
Peer pointers (2): *finger tables*

Finger Table at N80

<table>
<thead>
<tr>
<th>i</th>
<th>ft[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>5</td>
<td>112</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
</tbody>
</table>

Say $m=7$

ith entry at peer with id $n$ is first peer with id $\geq n + 2^i \pmod{2^m}$
What about the files?

• Filenames also mapped using same consistent hash function
  – SHA-1(filename) → 160 bit string (key), truncate to m
  – File is stored at first peer with id greater than its key (mod $2^m$)

• File cnn.com/index.html that maps to key K42 is stored at first peer with id greater than 42
  – If you store webpages this way, it’s called cooperative web caching (~ Memcached architecture)
  – Generic though
Mapping Files

Say $m=7$

File with key $K_{42}$ stored here
Say $m=7$

Who has \texttt{cnn.com/index.html}?
(hashes to K42)

File \texttt{cnn.com/index.html}
with key K42 stored here
At node $n$, send query for key $k$ to largest successor/finger entry $\leq k$ if none exist, send query to $\text{successor}(n)$

Say $m=7$

Who has `cnn.com/index.html`?
(hashes to K42)

File `cnn.com/index.html` with key K42 stored here
Search

At node $n$, send query for key $k$ to largest successor/finger entry $\leq k$
if none exist, send query to $\text{successor}(n)$

Say $m=7$

Who has \texttt{cnn.com/index.html}?
(hashes to K42)

File \texttt{cnn.com/index.html} with key K42 stored here
Search takes $O(\log(N))$ time

**Proof**

- (intuition): *at each step, distance between query and peer-with-file reduces by a factor of at least 2* (why?)

  Takes at most $m$ steps: $2^m$ is at most a constant multiplicative factor above $N$, lookup is $O(\log(N))$

- (intuition): after $\log(N)$ forwardings, distance to key is at most $2^m / N$ (why?)

  Number of node identifiers in a range of $2^m / N$ is $O(\log(N))$ with high probability (why? SHA-1!)

  So using *successors* in that range will be ok
Analysis (contd.)

- $O(\log(N))$ search time holds for file insertions too (in general for routing to any key)
  - “Routing” can thus be used as a building block for
    - All operations: insert, lookup, delete
- $O(\log(N))$ time true only if finger and successor entries correct
- When might these entries be wrong?
  - When you have failures
Search under peer failures

Say $m = 7$

Lookup fails
(N16 does not know N45)

Who has cnn.com/index.html?
(hashes to K42)

File cnn.com/index.html with key K42 stored here
Search under peer failures

One solution: maintain $r$ multiple successor entries
In case of failure, use successor entries

Say $m=7$

File cnn.com/index.html with key K42 stored here
Search under peer failures

• Choosing $r=2\log(N)$ suffices to maintain 
  *lookup correctness* w.h.p.
  
  – Say 50% of nodes fail
  
  – $\Pr$(at given node, at least one successor alive) =
    \[1 - \left(\frac{1}{2}\right)^{2\log N} = 1 - \frac{1}{N^2}\]
  
  – $\Pr$(above is true at all alive nodes) =
    \[\left(1 - \frac{1}{N^2}\right)^{N/2} = e^{-\frac{1}{2N}} \approx 1\]
Search under peer failures (2)

Say $m=7$

Who has `cnn.com/index.html`?

(hashes to K42)

Lookup fails (N45 is dead)

File `cnn.com/index.html` with key K42 stored here
Search under peer failures (2)

One solution: replicate file/key at $r$ successors and predecessors

Say $m=7$

Who has cnn.com/index.html? (hashes to K42)

K42 replicated

File cnn.com/index.html with key K42 stored here
Need to deal with dynamic changes

✓ Peers fail
• New peers join
• Peers leave
  – P2P systems have a high rate of *churn* (node join, leave and failure)
    • 25% per hour in Overnet (eDonkey)
    • 100% per hour in Gnutella
    • Lower in managed clusters, e.g., CSIL
    • Common feature in all distributed systems, including clouds

So, all the time, need to:
→ Update *successors* and *fingers*, and copy keys
New peers joining

Introducer directs N40 to N45 by routing to K40
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it
*N40 periodically talks to its neighbors to update finger table*

Say $m=7$
New peers joining (2)

N40 may need to copy some files/keys from N45
(files with fileid between 32 and 40)

Say $m=7$
New peers joining (3)

- A new peer affects $O(\log(N))$ other finger entries in the system, on average [Why?]
- Number of messages per peer join = $O(\log(N) \times \log(N))$

- Similar set of operations for dealing with peers leaving
  - For dealing with failures, need to couple above mechanisms with failure detectors
Experimental Results

• Sigcomm 01 paper had results from simulation of a C++ prototype
• SOSP 01 paper had more results from a 12-node Internet testbed deployment
• We’ll touch briefly on the first set
• 10000 peer system
Lookups

Average Messages per Lookup vs. Number of Nodes

log, as expected
Fault-tolerance

500 nodes (avg. path len=5)
Stabilization runs every 30 s
1 joins&fails every 10 s
(3 fails/stabilization round)
=> 6% lookups fail
Wrap-up Notes

- Memory: $O(\log(N))$ successor pointer, $m$ finger entries
- Indirection: store a pointer instead of the actual file
- Does not handle partitions (can you suggest a possible solution?)
Summary of Chord

- **Chord protocol**
  - More structured than Gnutella
  - $O(\log(N))$ memory and lookup cost
  - Simple lookup algorithm, rest of protocol complicated
  - Stabilization works, but how far can it go?
Wrap-up Notes

Applies to all p2p systems

• How does a peer join the system
  – Send an http request to well-known url for that P2P service - http://www.myp2pservice.com
  – Message routed (after DNS lookup) to a well known server which then initializes new peers’ neighbor table
  – Server only maintains a partial list of online clients

• Lookups can be speeded up by having each peer cache:
  – Queries and their results that it sees
  – All directory entries (filename,host) mappings that it sees
  – The files that pass through it
Up Next

• Blue Waters tour tomorrow
  – Please arrive on time at Blue Waters for your tour
  – Tours will not wait for latecomers (and you cannot change to a different tour than you’ve signed up)

• Reading: Sections 15.2, 15.3

• MP2: by now you should have started coding the failure detector
Optional Slides
Stabilization Protocol

- Concurrent peer joins, leaves, failures might cause loopiness of pointers, and failure of lookups
  - Chord peers periodically run a stabilization algorithm that checks and updates pointers and keys
  - Ensures non-loopiness of fingers, eventual success of lookups and $O(\log(N))$ lookups w.h.p.
  - [TechReport on Chord webpage] defines weak and strong notions of stability
  - Each stabilization round at a peer involves a constant number of messages
  - Strong stability takes $O(N^2)$ stabilization rounds (!)