Network Security
mod-net-sec

Lecture 7
Peer-to-peer networks-II (Chord)

Shishir Nagaraja
Jan 29, 2013
Acknowledgements

• The slides during this semester are based on ideas and material from the following sources:
  – Slides from Professor S. Gosh’s course at University of Iowa.
Plan for today

- P2P networks
  - Review Gnutella
  - Chord
FastTrack (KaZaA)

- Unstructured Peer-to-Peer System
- Hybrid between Gnutella and Napster
- Takes advantage of “healthier” participants in the system (higher bandwidth nodes, nodes that are around most of the time, nodes that are not freeloaders, etc.)
- Underlying technology in Kazaa, KazaaLite, Grokster
- Proprietary protocol, but some details available
- Like Gnutella, but with some peers designated as supernodes
A FastTrack-like System

Peers

Super-node
FastTrack (contd.)

- A supernode stores a directory listing \(<\text{filename}, \text{peer pointer}>\), similar to Napster servers.
- Supernode membership changes over time.
- Any peer can become (and stay) a supernode, provided it has earned enough reputation.
  - Kazaalite: participation level of a user between 0 and 1000, initially 10, then affected by length of periods of connectivity and total number of uploads from that client.
- A peer searches for a file by contacting a nearby supernode.
Final Comments on Unstructured P2P Systems

• How does a peer join the system (bootstrap)
  – Send an http request to well known URL
    http://www.myp2pservice.com
  – Message routed after DNS lookup to a well known server that has partial list of recently joined peers, and uses this to initialize new peers’ neighbor table

• 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 themselves
## 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>$O(1)$@client, $O(N)$@server</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Gnutella</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>
Hash Table (Phone Book)

<table>
<thead>
<tr>
<th>Keys</th>
<th>Indexes</th>
<th>Key-value pairs (records)</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>0</td>
<td>Lisa Smith +1-555-8976</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>John Smith +1-555-1234</td>
</tr>
<tr>
<td>Lisa Smith</td>
<td>872</td>
<td></td>
</tr>
<tr>
<td></td>
<td>873</td>
<td></td>
</tr>
<tr>
<td>Sam Doe</td>
<td>998</td>
<td>Sam Doe +1-555-5030</td>
</tr>
<tr>
<td></td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>
Hash Functions

- Fox: Hash function → DFCD3454
- The red fox *runs* across the ice: Hash function → 52ED879E
- The red fox *walks* across the ice: Hash function → 46042841
DHT=Distributed Hash Table

- **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)
- **Performance Concerns:**
  - Load balancing
  - Fault-tolerance
  - Efficiency of lookups and inserts
  - Locality
DHTs

• DHT research was motivated by Napster and Gnutella

• First four DHTs appeared in 2001
  – CAN
  – Chord
  – Pastry
  – Tapestry
  – BitTorrent

• 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>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)
Base Chord Protocol

• Uses concepts such as
  – Consistent hashing
  – Scalable key lookup
  – Handling of node dynamics
  – Stabilization protocol
Consistent Hashing

- **Consistent Hashing** on peer’s address
  - SHA-1 = Simple Hash Algorithm-1, a standard hashing function
    - (In 2005 security flaws were identified in SHA-1)
  - SHA-1(ip_address, port) → 160 bit string
    - Output truncated to $m \ (< 160)$ bits
    - Called **peer id** (integer between 0 and $2^m - 1$)
      - Example: SHA-1(140.45.3.10, 1024) = 45 (0101101) with $m = 7$
    - Not unique but **peer id** conflicts very very unlikely
  - Any node $A$ can calculate the **peer id** of any other node $B$, given $B$’s IP address and port number
  - We can then map peers to one of $2^m$ logical points on a circle
Ring of Peers

Example: \( m=7 \) (number of logical nodes is 128) = \( 2^7 \)

\( N=6 \) peers/nodes (\( N \) - number of physical nodes)
Peer pointers (1): *successors*

Say $m=7$

SHA-1(140.45.3.12, 1245) = 42 (0101010)

Each physical node maintains a successor pointer

42 stored in the successor N45

(similarly predecessors)
Peer pointers (2): *finger tables* (Scalable Key Location)

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>

Each node maintains a finger table

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

- Filenames are also mapped using **same consistent hash function**
  - SHA-1(filename) → 160 bits, truncated to \( m \) bits = file id or key
  - Assume \( K \) keys

- Example:
  - **File cnn.com/index.html** that maps to file id /key 42 is stored at first peer with id >= 42

- Note that we are considering a different file-sharing application here: **cooperative web caching**
  - “Peer” = client browser, “files” = html objects
  - Peers can now fetch html objects from other peers that have them, rather than from server
  - The same discussion applies to any other file sharing application, including that of mp3 files
Mapping Files

Say $m=7$

File `cnn.com/index.html` with key `K42` stored here
Say $m=7$

**Search**

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 $< k$
(all modulo $m$)
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 < \( k \) (all mod \( m \))

if none exist, send query to \textit{successor}(n)

Say \( m=7 \)

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

File \texttt{cnn.com/index.html} with key \texttt{K42} stored here
Analysis

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
  
  * 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 other applications than file sharing [can you name one?]
- $O(\log(N))$ time true only if finger and successor entries *correct*
- When might these entries be wrong?
Analysis (contd.)

• $O(\log(N))$ search time holds for file insertions too (in general for \textit{routing to any key})
  – “Routing” can thus be used as a building block for other applications than file sharing [can you name one?]

• $O(\log(N))$ time true only if finger and successor entries \textbf{correct}

• When might these entries be wrong?
  – When you have failures
Stabilization Protocol

• Maintaining finger tables only is expensive in case of dynamic joint and leave nodes

• Chord therefore separates correctness from performance goals via stabilization protocols

• Basic stabilization protocol
  – Keep successor’s pointers correct!
  – Then use them to correct finger tables
Search under peer failures

Lookup fails
(N16 does not know N45)

Say $m=7$

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

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

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

Say $m=7$

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

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

- Let $r$ be the successor list length
- Choosing $r = 2\log(N)$ suffices to maintain correctness with high probability
  - Say 50% of nodes fail
  - $\Pr($for given node, at least one successor alive$)=
    \begin{align*}
    1 - \left(\frac{1}{2}\right)^r &= 1 - \left(\frac{1}{2}\right)^{2\log N} = 1 - \frac{1}{N^2} \\
    \Pr($above is true for all alive nodes$)&= (1 - \frac{1}{N^2})^{N/2} = e^{-\frac{1}{2N}} \approx 1
    \end{align*}$
Search under peer failures (2)

Say \( m = 7 \)

Lookup fails (N45 is dead)

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

File \( \text{cnn.com/index.html} \) with key \( \text{K42} \) stored here
Search under peer failures (2)

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

Say $m=7$

Who has \texttt{cnn.com/index.html}?

(hashes to K42)

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

✓ Peers fail
• New peers join
• Peers leave

All the time

→ Need to update successors and fingers, and ensure keys reside in the right places
New peers joining

1. N40 acquires that N45 is its successor
2. N45 updates its info about predecessor to be N40
3. N32 runs stabilizer and asks N45 for predecessor
4. N45 returns N40
5. N32 updates its info about successor to be N40
6. N32 notifies N40 to be its predecessor

N40 periodically talks to neighbors to update own finger table

Peers also keep info about their predecessors to deal with dynamics

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
- Consider the number of messages to re-establish the Chord routing invariants and finger tables
- Number of messages per peer join = $O(\log(N) \times \log(N))$
- Similar set of operations for dealing with peers leaving
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
  - [TechReport on Chord webpage] defines weak and strong stability
  - Each stabilization round at a peer involves a constant number of messages
  - Strong stability takes $O(N^2)$ stabilization rounds (!)
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 of results
• 10,000 peer system
Lookups

(X-axis is logarithmic)

Number of Nodes

Average Messages per Lookup

log, as expected
Discussion

• Memory: $O(\log(N))$ successor pointer, $m$ finger entries
• Indirection: store a pointer instead of the actual file
• Does not handle partitions of the group. Possible solutions?
Discussion (2)

• When nodes are constantly joining, leaving, failing
  - Significant effect to consider: traces from the Overnet system show *hourly peer turnover rates* (*churn*) could be 10-15% of total number of nodes in system
  - Leads to *excessive (unnecessary) key copying*
    • further, remember that keys are replicated, so all these copies will need to be copied around as nodes join and leave
  - Stabilization algorithm may need to *consume more bandwidth* to keep up
  - There exist *alternative DHTs* that are churn-resistant
Discussion (3)

• Current status of project:
  – Protocol constantly undergoing change
  – File systems (CFS, Ivy) built on top of Chord
  – DNS lookup service built on top of Chord
  – Spawned research on many interesting issues about p2p systems

http://www.pdos.lcs.mit.edu/chord/
Summary

• Chord protocol
  – *Structured P2P*
  – $O(\log(N))$ memory and lookup cost
  – Simple lookup algorithm, rest of protocol complicated
  – Stabilization works, but how far can it go?