An Overview of Peer-to-Peer

Sami Rollins
Outline

• P2P Overview
  – What is a peer?
  – Example applications
  – Benefits of P2P
  – Is this just distributed computing?

• P2P Challenges

• Distributed Hash Tables (DHTs)
What is Peer-to-Peer (P2P)?

- Napster?
- Gnutella?
- Most people think of P2P as music sharing
What is a peer?

- Contrasted with Client-Server model
- Servers are centrally maintained and administered
- Client has fewer *resources* than a server
What is a peer?

• A peer’s resources are similar to the resources of the other participants
• P2P – peers communicating directly with other peers and sharing resources
• Often administered by different entities
  – Compare with DNS
P2P Application Taxonomy

P2P Systems

- Distributed Computing
  - SETI@home
- File Sharing
  - Gnutella
- Collaboration
  - Jabber
- Platforms
  - JXTA
Distributed Computing
Collaboration

(sendMessage) → (receiveMessage) → (sendMessage)
Collaboration

sendMessage  receiveMessage  sendMessage  receiveMessage
# Platforms

<table>
<thead>
<tr>
<th>Gnutella</th>
<th>Instant Messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Peers</td>
<td>…</td>
</tr>
</tbody>
</table>
P2P Goals/Benefits

• Cost sharing
• Resource aggregation
• Improved scalability/reliability
• Increased autonomy
• Anonymity/privacy
• Dynamism
• Ad-hoc communication
P2P File Sharing

• Centralized
  – Napster

• Decentralized
  – Gnutella

• Hierarchical
  – Kazaa

• Incentivized
  – BitTorrent

• Distributed Hash Tables
  – Chord, CAN, Tapestry, Pastry
Challenges

• Peer discovery
• Group management
• Search
• Download
• Incentives
Metrics

- Per-node state
- Bandwidth usage
- Search time
- Fault tolerance/resiliency
Centralized

• Napster model
  – Server contacted during search
  – Peers directly exchange content

• Benefits:
  – Efficient search
  – Limited bandwidth usage
  – No per-node state

• Drawbacks:
  – Central point of failure
  – Limited scale
Decentralized (Flooding)

- **Gnutella model**
  - Search is flooded to neighbors
  - Neighbors are determined *randomly*

- **Benefits:**
  - No central point of failure
  - Limited per-node state

- **Drawbacks:**
  - Slow searches
  - Bandwidth intensive
Hierarchical

- Kazaa/new Gnutella model
  - Nodes with high bandwidth/long uptime become *supernodes/ultrapeers*
  - Search requests sent to supernode
  - Supernode caches info about attached leaf nodes
  - Supernodes connect to each other (32 in Limewire)

- Benefits:
  - Search faster than flooding

- Drawbacks:
  - Many of the same problems as decentralized
  - Reconfiguration when supernode fails
1. Download torrent

2. Get list of peers and seeds (the swarm)

3. Exchange vector of content downloaded with peers

4. Exchange content w/ peers

5. Update w/ progress
BitTorrent

• Key Ideas
  – Break large files into small blocks and download blocks individually
  – Provide incentives for uploading content
    • Allow download from peers that provide best upload rate

• Benefits
  – Incentives
  – Centralized search
  – No neighbor state (except the peers in your swarm)

• Drawbacks
  – “Centralized” search
    • No central repository
Distributed Hash Tables (DHT)

• Chord, CAN, Tapestry, Pastry model
  – AKA Structured P2P networks
  – Provide performance guarantees
  – If content exists, it will be found

• Benefits:
  – More efficient searching
  – Limited per-node state

• Drawbacks:
  – Limited fault-tolerance vs redundancy
DHTs: Overview

• Goal: Map key to value
• Decentralized with bounded number of neighbors
• Provide guaranteed performance for search
  – If content is in network, it will be found
  – Number of messages required for search is bounded
• Provide guaranteed performance for join/leave
  – Minimal number of nodes affected
• Suitable for applications like file systems that require guaranteed performance
Comparing DHTs

- Neighbor state
- Search performance
- Join algorithm
- Failure recovery
CAN

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

• Goals
  – Scales to hundreds of thousands of nodes
  – Handles rapid arrival and failure of nodes

• Properties
  – Routing table size $O(d)$
  – Guarantees that a file is found in at most $d^*n^{1/d}$ steps, where $n$ is the total number of nodes

Slide modified from another presentation
CAN Example: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
  - Node n1:(1, 2) first node that joins \(\rightarrow\) cover the entire space

Slide modified from another presentation
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins
- n2 contacts n1
- n1 splits its area and assigns half to n2
CAN Example: Two Dimensional Space

- Nodes n3:(3, 5) n4:(5, 5) and n5:(6,6) join
- Each new node sends JOIN request to an existing node chosen randomly
- New node gets neighbor table from existing node
- New and existing nodes update neighbor tables and neighbors accordingly
  - before n5 joins, n4 has neighbors n2 and n3
  - n5 adds n4 and n2 to neighborlist
  - n2 updated to include n5 in neighborlist
- Only $O(2^d)$ nodes are affected
CAN Example: Two Dimensional Space

- Bootstrapping - assume CAN has an associated DNS domain and domain resolves to IP of one or more bootstrap nodes
- Optimizations - landmark routing
  - Ping a landmark server(s) and choose an existing node based on distance to landmark
CAN Example: Two Dimensional Space

- Nodes: n1:(1, 2); n2:(4,2); n3:(3, 5); n4:(5,5); n5:(6,6)
- Items: f1:(2,3); f2:(5,1); f3:(2,1); f4:(7,5);
CAN Example: Two Dimensional Space

• Each item is stored by the node who owns its mapping in the space

Slide modified from another presentation
CAN: Query Example

- Forward query to the neighbor that is closest to the query *id* (Euclidean distance)
- Example: assume n1 queries f4
CAN: Query Example

- Forward query to the neighbor that is closest to the query id
- Example: assume n1 queries f4
CAN: Query Example

- Forward query to the neighbor that is closest to the query id
- Example: assume n1 queries f4

Slide modified from another presentation
CAN: Query Example

- Content guaranteed to be found in $d \times n^{1/d}$ hops
  - Each dimension has $n^{1/d}$ nodes
- Increasing the number of dimensions reduces path length but increases number of neighbors

Slide modified from another presentation
Node Failure Recovery

• Detection
  – Nodes periodically send refresh messages to neighbors

• Simple failures
  – neighbor’s neighbors are cached
  – when a node fails, one of its neighbors takes over its zone
    • when a node fails to receive a refresh from neighbor, it sets a timer
    • many neighbors may simultaneously set their timers
    • when a node’s timer goes off, it sends a TAKEOVER to the failed node’s neighbors
    • when a node receives a TAKEOVER it either (a) cancels its timer if the zone volume of the sender is smaller than its own or (b) replies with a TAKEOVER
Chord

- Each node has m-bit id that is a SHA-1 hash of its IP address
- Nodes are arranged in a circle modulo m
- Data is hashed to an id in the same id space
- Node $n$ stores data with id between $n$ and $n$'s predecessor
  - 0 stores 4-0
  - 1 stores 1
  - 3 stores 2-3
Chord

- Simple query algorithm:
  - Node maintains successor
  - To find data with id $i$, query successor until successor > $i$ found

- Running time?
Chord

• In reality, nodes maintain a finger table with more routing information
  – For a node \( n \), the \( i^{th} \) entry in its finger table is the first node that succeeds \( n \) by at least \( 2^{i-1} \)
• Size of finger table?
Chord

• In reality, nodes maintain a *finger table* with more routing information
  – For a node $n$, the $i^{th}$ entry in its finger table is the first node that succeeds $n$ by at least $2^{i-1}$
• Size of finger table?
• $O(\log N)$
Chord

query:
hash key to get id
if id == node id - data found
else if id in finger table - data found
else
    p = find_predecessor(id)
    n = find_successor(p)

find_predecessor(id):
choose n in finger table closest to id
if n < id < find_successor(n)
    return n
else
    ask n for finger entry closest to id and
    recurse
Chord

• Running time of query algorithm?
  – Problem size is halved at each iteration
• Running time of query algorithm?
  – $O(\log N)$
Chord

• Join
  – initialize predecessor and fingers
  – update fingers and predecessors of existing nodes
  – transfer data
Chord

- Initialize predecessor and finger of new node \( n^* \)
  - \( n^* \) contacts existing node in network \( n \)
  - \( n \) does a lookup of predecessor of \( n^* \)
  - for each entry in finger table, look up successor
- Running time - \( O(m \log N) \)
- Optimization - initialize \( n^* \) with finger table of successor
  - with high probability, reduces running time to \( O(\log N) \)
Chord

• Update existing nodes
  – $n^*$ becomes $i$th finger of a node $p$ if
    • $p$ precedes $n$ by at least $2^{i-1}$
    • the $i$th finger of $p$ succeeds $n$
  – start at predecessor of $n^*$ and walk backwards
  – for $i=1$ to 3:
    • find predecessor of $n^*-2^{i-1}$
    • update table and recurse

• Running time $O(\log^2 N)$
Chord

- **Stabilization**
  - Goal: handle concurrent joins
  - Periodically, ask successor for its predecessor
  - If your successor’s predecessor isn’t you, update
  - Periodically, refresh finger tables

- **Failures**
  - keep list of $r$ successors
  - if successor fails, replace with next in the list
  - finger tables will be corrected by stabilization algorithm
DHTs – Tapestry/Pastry

- Global mesh
- Suffix-based routing
- Uses underlying network distance in constructing mesh
## Comparing Guarantees

<table>
<thead>
<tr>
<th>Model</th>
<th>Search</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chord</strong></td>
<td>log N</td>
<td>log N</td>
</tr>
<tr>
<td>Uni-dimensional</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CAN</strong></td>
<td>dN^{1/d}</td>
<td>2d</td>
</tr>
<tr>
<td>Multi-dimensional</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tapestry</strong></td>
<td>log_{b}N</td>
<td>b log_{b}N</td>
</tr>
<tr>
<td>Global Mesh</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pastry</strong></td>
<td>log_{b}N</td>
<td>b log_{b}N + b</td>
</tr>
<tr>
<td>Neighbor map</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remaining Problems?

- Hard to handle highly dynamic environments
- Usable services
- Methods don’t consider peer characteristics
Measurement Studies

- “Free Riding on Gnutella”
- Most studies focus on Gnutella
- Want to determine how users behave
- Recommendations for the best way to design systems
Free Riding Results

- Who is sharing what?
- August 2000

<table>
<thead>
<tr>
<th>The top</th>
<th>Share</th>
<th>As percent of whole</th>
</tr>
</thead>
<tbody>
<tr>
<td>333 hosts (1%)</td>
<td>1,142,645</td>
<td>37%</td>
</tr>
<tr>
<td>1,667 hosts (5%)</td>
<td>2,182,087</td>
<td>70%</td>
</tr>
<tr>
<td>3,334 hosts (10%)</td>
<td>2,692,082</td>
<td>87%</td>
</tr>
<tr>
<td>5,000 hosts (15%)</td>
<td>2,928,905</td>
<td>94%</td>
</tr>
<tr>
<td>6,667 hosts (20%)</td>
<td>3,037,232</td>
<td>98%</td>
</tr>
<tr>
<td>8,333 hosts (25%)</td>
<td>3,082,572</td>
<td>99%</td>
</tr>
</tbody>
</table>
Saroiu et al Study

- How many peers are server-like...client-like?
  - Bandwidth, latency
- Connectivity
- Who is sharing what?
Saroiu et al Study

- May 2001
- Napster crawl
  - query index server and keep track of results
  - query about returned peers
  - don’t capture users sharing unpopular content
- Gnutella crawl
  - send out ping messages with large TTL
Results Overview

• Lots of heterogeneity between *peers*
  – Systems should consider peer capabilities
• Peers lie
  – Systems must be able to verify reported peer capabilities or measure true capabilities
Measured Bandwidth

Figure 3. Left: CDFs of upstream and downstream bottleneck bandwidths for Gnutella peers; Right: CDFs of downstream bottleneck bandwidths for Napster and Gnutella peers.
Figure 4. Left: Reported bandwidths for Napster peers; Right: Reported bandwidths for Napster peers, excluding peers that reported “unknown”.
Measured Latency

Figure 5. Left: Measured latencies to Gnutella peers; Right: Correlation between Gnutella peers’ downstream bottleneck bandwidth and latency.
Figure 6. IP-level uptime of peers (“Internet Host Uptime”), and application-level uptime of peers (“Gnutella/Napster Host Uptime”) in both Napster and Gnutella, as measured by the percentage of time the peers are reachable.

Figure 7. The distribution of Napster/Gnutella session durations.
Number of Shared Files

Figure 8. Left: The number of shared files for Gnutella peers; Right: The number of shared files for Napster and Gnutella peers (peers with no files to share are excluded).
Figure 15. Left: Topology of the Gnutella network as of February 16, 2001 (1771 peers); Middle: Topology of the Gnutella network after a random 30% of the nodes are removed; Right: Topology of the Gnutella network after the highest-degree 4% of the nodes are removed.
Points of Discussion

• Is it all hype?
• Should P2P be a research area?
• Do P2P applications/systems have common research questions?
• What are the “killer apps” for P2P systems?
Conclusion

• P2P is an interesting and useful model
• There are lots of technical challenges to be solved