P2P: Distributed Hash Tables

Chord + Routing Geometries

Nirvan Tyagi
CS 6410 Fall16
Peer-to-peer (P2P)
Peer-to-peer (P2P)

Decentralized!

Hard to coordinate with peers joining and leaving
Peer-to-peer (P2P)
Napster (1999)
Peer-to-peer (P2P)
Napster (1999)
Peer-to-peer (P2P)

Napster (1999)

Problem - Centralized index server
Peer-to-peer (P2P)

Napster (1999)

Problem - Centralized index server

Solution - Distributed hash table
Distributed Hash Tables (DHT)

Hash table

<table>
<thead>
<tr>
<th>k_0</th>
<th>v_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_1</td>
<td>v_1</td>
</tr>
<tr>
<td>k_2</td>
<td>v_2</td>
</tr>
<tr>
<td>k_3</td>
<td>v_3</td>
</tr>
<tr>
<td>k_4</td>
<td>v_4</td>
</tr>
<tr>
<td>k_5</td>
<td>v_5</td>
</tr>
</tbody>
</table>
Distributed Hash Tables (DHT)

Hash table

<table>
<thead>
<tr>
<th>k_0</th>
<th>v_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_1</td>
<td>v_1</td>
</tr>
<tr>
<td>k_2</td>
<td>v_2</td>
</tr>
<tr>
<td>k_3</td>
<td>v_3</td>
</tr>
<tr>
<td>k_4</td>
<td>v_4</td>
</tr>
<tr>
<td>k_5</td>
<td>v_5</td>
</tr>
</tbody>
</table>
Distributed Hash Tables (DHT)

Desirable Properties

- Decentralization
- Load-balancing
- Scalability
- Availability
Outline

Chord
- Specific DHT protocol for P2P systems
- Simple, efficient

DHT Routing Geometry
- Effect of different DHT protocols on desirable system properties
Chord
A scalable P2P lookup service for internet applications
Ion Stoica, Robert Morris, David Karger
Frans Kaashoek, Hari Balakrishnan
Chord - Overview

How to assign keys to peers?
Chord - Overview

Hash table

<table>
<thead>
<tr>
<th>$k_0$</th>
<th>$v_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$v_4$</td>
</tr>
<tr>
<td>$k_5$</td>
<td>$v_5$</td>
</tr>
</tbody>
</table>
Chord - Overview

Hash table

<table>
<thead>
<tr>
<th>k_0</th>
<th>v_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_1</td>
<td>v_1</td>
</tr>
<tr>
<td>k_2</td>
<td>v_2</td>
</tr>
<tr>
<td>k_3</td>
<td>v_3</td>
</tr>
<tr>
<td>k_4</td>
<td>v_4</td>
</tr>
<tr>
<td>k_5</td>
<td>v_5</td>
</tr>
</tbody>
</table>
Chord - Overview

Hash table

<table>
<thead>
<tr>
<th>$k_0$</th>
<th>$v_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$v_4$</td>
</tr>
<tr>
<td>$k_5$</td>
<td>$v_5$</td>
</tr>
</tbody>
</table>
Chord - Overview

Identifier ring over hash space $2^m$
Chord - Overview

Identifier ring over hash space \(2^m\)

- Node id = hash( node )
- Key id = hash( key )
Chord - Overview

Identifier ring over hash space $2^m$

- Node id = hash(node)
- Key id = hash(key)
- Successor(id)

Finger table for node at id $i$

<table>
<thead>
<tr>
<th>finger</th>
<th>node id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>succ($i$)</td>
</tr>
<tr>
<td>2</td>
<td>succ($i + 2$)</td>
</tr>
<tr>
<td>$j$</td>
<td>succ($i + 2^{j-1}$)</td>
</tr>
</tbody>
</table>
Chord - Overview

Identifier ring

0

8

12

0 = node

4 = key
Chord - Overview

Identifier ring

0

4

8

12

= node

= key
Chord - Overview

Identifier ring

= node
= key

<table>
<thead>
<tr>
<th>finger</th>
<th>node id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>succ(i)</td>
</tr>
<tr>
<td>2</td>
<td>succ(i + 2)</td>
</tr>
<tr>
<td>3</td>
<td>succ(i + 2^2)</td>
</tr>
<tr>
<td>4</td>
<td>succ(i + 2^3)</td>
</tr>
</tbody>
</table>
Chord - Overview

Identifier ring

<table>
<thead>
<tr>
<th>finger</th>
<th>node id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>succ(4)</td>
</tr>
<tr>
<td>2</td>
<td>succ(4 + 2)</td>
</tr>
<tr>
<td>3</td>
<td>succ(4 + 2^2)</td>
</tr>
<tr>
<td>4</td>
<td>succ(4 + 2^3)</td>
</tr>
</tbody>
</table>
Chord - Overview

Identifier ring

0

12

8

4

= node

= key

<table>
<thead>
<tr>
<th>finger</th>
<th>node id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Chord - Lookup

Identifier ring

<table>
<thead>
<tr>
<th>finger</th>
<th>node id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

find_successor(id):
  p = find_predecessor(id)
  return p.successor

find_predecessor(id):
  n = self
  while id not between (n, n.successor):
    n = n.closest_preceding_finger(id)
  return n
Chord - Lookup

Identifier ring

```
find_successor(id):
p = find_predecessor(id)
return p.successor

find_predecessor(id):
n = self
while id not between (n, n.successor):
    n = n.closest_preceding_finger(id)
return n
```

lookup(10)
Chord - Lookup

Identifier ring

find_successor(id):
  p = find_predecessor(id)
  return p.successor

find_predecessor(id):
  n = self
  while id not between (n, n.successor):
    n = n.closest_preceding_finger(id)
  return n

lookup(10)

follow finger 3 to node id 8
Chord - Lookup

Identifier ring

find_successor(id):
p = find_predecessor(id)
return p.successor

find_predecessor(id):
n = self
while id not between (n, n.successor]:
    n = n.closest_preceding_finger(id)
return n

lookup(10)
follow finger 3 to node id 8
node id 8 identifies as predecessor of id 10
Chord - Lookup

Identifier ring

find_successor(id):
  p = find_predecessor(id)
  return p.successor

find_predecessor(id):
  n = self
  while id not between (n, n.successor):
    n = n.closest_preceding_finger(id)
  return n

lookup(10)

follow finger 3 to node id 8

node id 8 identifies as predecessor of id 10

complete lookup at successor of node id 8
**Chord - Lookup**

**Identifier ring**

**find_successor(id):**
```
p = find_predecessor(id)
return p.successor
```

**find_predecessor(id):**
```
n = self
while id not between (n, n.successor):
    n = n.closest_preceding_finger(id)
return n
```

**lookup(10):**

follow finger 3 to node id 8

node id 8 identifies as predecessor of id 10

complete lookup at successor of node id 8

**Hops?**
Each finger lookup halves distance to key 
$O(\log N)$

---

**Table:**

<table>
<thead>
<tr>
<th>finger</th>
<th>node id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Chord - Joins + Stabilization

Identifier ring

join():
  self.predecessor = null
  self.successor = find_successor(self)

stabilize():
  p = self.successor.predecessor
  if p between (self, self.successor):
    self.successor = p
    self.successor.notify(self)

notify(n):
  if self.predecessor == null ||
    n between (self.predecessor, self):
    self.predecessor = n
Chord - Joins + Stabilization

Identifier ring

join():
  self.predecessor = null
  self.successor = find_successor(self)

stabilize():
  p = self.successor.predecessor
  if p between (self, self.successor):
    self.successor = p
    self.successor.notify(self)

notify(n):
  if self.predecessor == null ||
    n between (self.predecessor, self):
    self.predecessor = n

predecessor = 4
successor = 8

predecessor = 5
successor = 11
Chord - Joins + Stabilization

Identifier ring

join():
    self.predecessor = null
    self.successor = find_successor(self)

stabilize():
    p = self.successor.predecessor
    if p between (self, self.successor):
        self.successor = p
        self.successor.notify(self)

notify(n):
    if self.predecessor == null ||
        n between (self.predecessor, self):
        self.predecessor = n
Chord - Joins + Stabilization

join():
    self.predecessor = null
    self.successor = find_successor(self)

stabilize():
    p = self.successor.predecessor
    if p between (self, self.successor):
        self.successor = p
        self.successor.notify(self)

notify(n):
    if self.predecessor == null ||
    n between (self.predecessor, self):
        self.predecessor = n

Identifier ring

0

4

12

predecessor = 4
successor = 8

stabilize()
### Chord - Joins + Stabilization

**Join**: 
- `self.predecessor = null` 
- `self.successor = find_successor(self)`

**Stabilize**: 
- `p = self.successor.predecessor` 
- If `p` is between (`self`, `self.successor`): 
  - `self.successor = p` 
  - `self.successor.notify(self)`

**Notify** (`n`): 
- If `self.predecessor == null || n` is between (`self.predecessor`, `self`): 
  - `self.predecessor = n` 

---

**Diagram**:
1. **Identifier ring**: 
   - Node `0`: 
     - `predecessor = 4` 
     - `successor = 11`
   - Node `4`: 
     - `predecessor = 4` 
     - `successor = 8`
   - Node `8`: 
     - `predecessor = null` 
     - `successor = 5`
Chord - Joins + Stabilization

join():
    self.predecessor = null
    self.successor = find_successor(self)

stabilize():
    p = self.successor.predecessor
    if p between (self, self.successor):
        self.successor = p
        self.successor.notify(self)

notify(n):
    if self.predecessor == null ||
        n between (self.predecessor, self):
        self.predecessor = n

Outcomes of incomplete stabilization:
1. Lookup unaffected
2. Fingers out-dated, successors correct -> lookup slow but correct
3. Successors in lookup region still stabilizing -> lookup fails
Chord - Failure + Replication

Identifier ring

Maintain list of k successors
Keys replicated on all k successors

predecessor = [6, 5]
successor = [11, 1]

predecessor = [4, 1]
successor = [6, 8]

predecessor = [5, 4]
successor = [8, 11]
Load balance

Failure resilience

Lookup path length

Lookup latency
Thoughts on Chord performance?
Load balance

Lookup path length

Failure resilience

Lookup latency

Improvements to Chord routing and failure resilience in future works Pastry + Bamboo
DHT Routing Geometries

Ring (Chord)

Tree (Tapestry, PRR)

Hypercube (CAN)

Butterfly (Viceroy)

XOR (Kademlia)

Hybrid (Pastry, Bamboo)
Proximity + Resilience

Proximity - Pick routes through “physically nearby” peers, reducing latency

Resilience - Continue to route requests despite network churn and failure
Proximity + Resilience

Proximity - Pick routes through “physically nearby” peers, reducing latency
Resilience - Continue to route requests despite network churn and failure

Flexibility

Neighbor selection - options in selecting which peers to keep in routing table
Route selection - options in selecting where to route to given a destination
Proximity + Resilience

Proximity - Pick routes through “physically nearby” peers, reducing latency

Resilience - Continue to route requests despite network churn and failure

Flexibility

Neighbor selection - options in selecting which peers to keep in routing table

Route selection - options in selecting where to route to given a destination

Flexibility in neighbor selection -> good proximity

Flexibility in route selection -> good resilience
DHT Routing Geometries

**Ring** (Chord)

**Tree** (Tapestry, PRR)

**Hypercube** (CAN)

Butterfly (Viceroy)

XOR (Kademlia)

Hybrid (Pastry, Bamboo)
DHT Routing Geometries - Tree

XXX

0XX

00X

000 001 010 011 100 101 110 111
DHT Routing Geometries - Tree

Neighbor selection - one neighbor for each prefix in opposite subtree
DHT Routing Geometries - Tree

Neighbor selection - one neighbor for each prefix in opposite subtree
Neighbor selection - one neighbor for each prefix in opposite subtree

Route selection - route to neighbor in subtree of destination
DHT Routing Geometries - Tree

Neighbor selection - one neighbor for each prefix in opposite subtree

Route selection - route to neighbor in subtree of destination
DHT Routing Geometries - Tree

Neighbor selection - one neighbor for each prefix in opposite subtree

Route selection - route to neighbor in subtree of destination

Good flexibility in neighbor selection
Poor flexibility in route selection
DHT Routing Geometries - Hypercube
DHT Routing Geometries - Hypercube

Neighbor selection - neighbor differs in the bit of one dimension
DHT Routing Geometries - Hypercube

Neighbor selection - neighbor differs in the bit of one dimension
DHT Routing Geometries - Hypercube

Neighbor selection - neighbor differs in the bit of one dimension

Route selection - route to destination by correcting any differing bit
DHT Routing Geometries - Hypercube

Neighbor selection - neighbor differs in the bit of one dimension

Route selection - route to destination by correcting any differing bit
DHT Routing Geometries - Hypercube

Neighbor selection - neighbor differs in the bit of one dimension

Route selection - route to destination by correcting any differing bit

Poor flexibility in neighbor selection

Good flexibility in route selection
DHT Routing Geometries - Ring
DHT Routing Geometries - Ring

Neighbor selection - one neighbor in each finger interval
DHT Routing Geometries - Ring

Neighbor selection - one neighbor in each finger interval
DHT Routing Geometries - Ring

Neighbor selection - one neighbor in each finger interval

Route selection - route to destination by making progress along ring
DHT Routing Geometries - Ring

Neighbor selection - one neighbor in each finger interval

Route selection - route to destination by making progress along ring
DHT Routing Geometries - Ring

Neighbor selection - one neighbor in each finger interval

Route selection - route to destination by making progress along ring

Good flexibility in neighbor selection

Good flexibility in route selection
## DHT Routing Geometries - Summary

<table>
<thead>
<tr>
<th></th>
<th>Tree</th>
<th>Hypercube</th>
<th>Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbor selection (Proximity)</td>
<td>✅</td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>Route selection (Resilience)</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>