Kademlia: A Peer-to-peer Information System Based on the XOR Metric

Petar Maymounkov and David Mazières
New York University
(Presented In 1th International Workshop on P2P Systems 2002)

Amir H. Payberah (amir@sics.se)
Seif Haridi (haridi@kth.se)
Core Idea
Definition

• Each object is stored at the $k$ closest nodes to the object's ID.

• **Distance** between $id_1$ and $id_2$: $d(id_1, id_2) = id_1 \ XOR \ id_2$
  
  - If ID space is 3 bits:

    $$d(1, 4) = d(001_2, 100_2)$$

    $$= 001_2 \ XOR \ 100_2$$

    $$= 101_2$$

    $$= 5$$
Core Idea - 1

- **Kbucket**: each node keeps a list of information for nodes of distance between $2^i$ and $2^{i+1}$.
Core Idea - 2

- Closest nodes in ID space
Core Idea - 3

Node

KBucket List

closest nodes to Q are stored here

... and select $\alpha$ nodes from the appropriate $kbucket$
Core Idea - 4

```
P
FIND_NODE(Q)

A
FIND_NODE(Q)

B
FIND_NODE(Q)

C
FIND_NODE(Q)
```
Core Idea - 5

A

Find $k$ closest nodes to $Q$

B

Find $k$ closest nodes to $Q$

C

Find $k$ closest nodes to $Q$
Core Idea - 6

Diagram:
- Node P
  - Connects to A
    - Returns k closest nodes to Q
  - Connects to B
    - Returns k closest nodes to Q
  - Connects to C
    - Returns k closest nodes to Q
When $P$ receives any message from another node, it updates the appropriate kbucket for the sender’s node ID.

Received information from A, B and C

... again select $\alpha$ nodes from the received information
Core Idea - 9

Repeats this procedure iteratively until received information in round $n-1$ and $n$ are the same.
P resends the FIND_NODE to $k$ closest nodes it has not already queried...
Let's Look Inside of Kademlia
Node State

• **Kbucket**: each node keeps a list of information for nodes of distance between $2^i$ and $2^{i+1}$.
  
  - $0 \leq i < 160$
  
  - Sorted by time last seen.
Node State

• **Kbucket:** each node keeps a list of information for nodes of distance between $2^i$ and $2^{i+1}$.
  
  - $0 \leq i < 160$
  - Sorted by time last seen.

<table>
<thead>
<tr>
<th>111</th>
<th>101</th>
<th>100</th>
<th>011</th>
<th>010</th>
<th>001</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>[1, 2) - Two first bits in common</td>
<td>[2, 4) - First bit in common</td>
<td>[4, 8) - No common prefix</td>
<td></td>
</tr>
</tbody>
</table>
Kademlia RPCs

• **PING**
  ▪ Probes a node to see if it is online.

• **STORE**
  ▪ Instructs a node to store a <key, value> pair.

• **FIND_NODE**
  ▪ Returns information for the k nodes it knows about closest to the target ID.
  ▪ It can be from one kbucket or more.

• **FIND_VALUE**
  ▪ Like FIND_NODE, ...
  ▪ But if the recipient has stored they <key, value>, it just returns the stored value.
Store Data

• The <key, value> data is stored in $k$ closest nodes to the key.
Lookup Service

Step 1: 001

Step 2: 110

Step 3: 100
Maintaining Kbucket List (Routing Table)

• When a Kademlia node receives any message from another node, it updates the appropriate kbucket for the sender’s node ID.

• If the sending node already exists in the kbucket:
  ▪ Moves it to the tail of the list.

• Otherwise:
  ▪ If the bucket has fewer than k entries:
    • Inserts the new sender at the tail of the list.
  ▪ Otherwise:
    ▪ Pings the kbucket’s least-recently seen node:
      • If the least-recently seen node fails to respond:
        ▪ it is evicted from the k-bucket and the new sender inserted at the tail.
      ▪ Otherwise:
        ▪ it is moved to the tail of the list, and the new sender’s contact is discarded.
Maintaining Kbucket List (Routing Table)

• Buckets will generally be kept constantly fresh, due to traffic of requests travelling through nodes.

• When there is no traffic: each peer picks a random ID in kbucket's range and performs a node search for that ID.
Join

- Node $P$ contacts to an already participating node $Q$.
- $P$ inserts $Q$ into the appropriate kbucket.
- $P$ then performs a node lookup for its own node ID.
Leave And Failure

- No action!

- If a node does not respond to the PING message, remove it from the table.
Kademlia and other DHTs
Kademlia vs. Chord

• like Chord
  ▪ When $\alpha = 1$ the lookup algorithm resembles Chord's in term of message cost and the latency of detecting failed nodes.

• Unlike Chord
  ▪ XOR metric is symmetric, while Chord's metric is asymmetric.
# Kademlia vs. Pastry

- **like Pastry**
  - The same routing table.

- **Unlike Pastry**
  - $\alpha = 3$ by default in Kademlia, while $\alpha = 1$ in Pastry.

<table>
<thead>
<tr>
<th>Pastry</th>
<th>Node 001 routing table</th>
<th>Kademlia</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P = 2$</td>
<td>000</td>
<td>[1, 2)</td>
</tr>
<tr>
<td>$P = 1$</td>
<td>010 011</td>
<td>[2, 4)</td>
</tr>
<tr>
<td>$P = 0$</td>
<td>110 100 111 101</td>
<td>[4, 8)</td>
</tr>
</tbody>
</table>
DONE!
A Page To Remember

Step 1: 001

Step 2: 110

Step 3: 100

Kademlia, 23rd March 2009
References

Question?