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

Based on slides by Amir H. Payberah (amir@sics.se)
Kademlia Basics

- Kademlia is a key-value (object) store.
- Each object is stored at the k closest nodes to the object's ID.
- **Distance** between id1 and id2: \( d(id1, id2) = id1 \ XOR \ id2 \)
  - If ID space is 3 bits:
    
    \[
    d(1, 4) = d(001_2, 100_2) \\
    = 001_2 \ XOR \ 100_2 \\
    = 101_2 \\
    = 5
    \]
Kademlia Routing Table

- **Kbucket**: each node keeps a list of references to nodes (contacts) of distance between $2^i$ and $2^{i+1}$ for $i=1$ to $i=N$.
- Each Kbucket has max $k$ entries.
Kademlia Tuning Parameters

• B is the size in bits of the keys used to identify nodes and store and retrieve data; in basic Kademlia this is 160, the length of an SHA1 digest (hash).
• k is the maximum number of contacts stored in a Kbucket; this is normally 20.
• alpha (\( \alpha \)) represents the degree of parallelism in network calls, usually 3.

• Other constants used in Kad:
  ▪ tExpire = 86400s, the time after which a key/value pair expires; this is a time-to-live (TTL) from the original publication date
  ▪ tRefresh = 3600s, after which an otherwise unaccessed bucket must be refreshed
  ▪ tReplicate = 3600s, the interval between Kademlia replication events, when a node is required to publish its entire database
  ▪ tRepublish = 86400s, the time after which the original publisher must republish a key/value pair
FIND_NODE in Kademlia

- Closest nodes in ID space
FIND_NODE in Kademlia

Lookup Q

Node P

KBucket List

closest nodes to Q are stored here

... and select $\alpha$ nodes from the appropriate kbucket
FIND_NODE in Kademlia
FIND_NODE in Kademlia

Find k closest nodes to Q

A

Find k closest nodes to Q

B

Find k closest nodes to Q

C
FIND_NODE in Kademlia

- Returns \( k \) closest nodes to Q
- P
- A
- Returns \( k \) closest nodes to Q
- Returns \( k \) closest nodes to Q
- B
- C
**FIND_NODE in Kademlia, Update Kbuckets**

When \( P \) receives a response from a node, it updates the appropriate Kbucket for the sender’s node ID.

\( P \) issues up to \( \alpha \) new requests to nodes it has not yet queried from the set of nodes received in the responses.
FIND_NODE in Kademlia

FIND_NODE(Q)

P

M

FIND_NODE(Q)

FIND_NODE(Q)

O

N
FIND_NODE in Kademlia

Received information in round $n-1$  [Diagram]

Received information in round $n$  [Diagram]

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 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.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>110</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>111</td>
<td>101</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td>111</td>
<td>101</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1, 2)</td>
<td>[2, 4)</td>
<td>[4, 8)</td>
<td></td>
</tr>
</tbody>
</table>
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>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>010</td>
<td>001</td>
</tr>
</tbody>
</table>

- [1, 2) - Two first bits in common
- [2, 4) - First bit in common
- [4, 8) - No common prefix
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.
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.
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 should 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 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 vs. Chord
Kademlia vs. Chord

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

• Unlike Chord
  ▪ XOR metric is symmetric, while Chord's metric is asymmetric.
Summary

Step 1: 001

Step 2: 110

Step 3: 100
References

• Kademlia Specification

• Petar Maymounkov and David Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", IPTPS '02

• Daniel Stutzbach and Reza Rejaie, "Improving Lookup Performance over a Widely-Deployed DHT", INFOCOM '06

• Raul Jimenez, Flutra Osmani and Bjorn Knutsson, “Sub-Second Lookups on a Large-Scale Kademlia-Based Overlay”, P2P '11.