Kademlia: A P2P Information System Based on the XOR Metric

Today
- By Petar Mayamounkov and David Mazières, presented at IPTPS 2002

Next
- Paper presentation and discussion
Peer-to-peer

- A distributed system architecture
  - No centralized control
  - Nodes are symmetric in function
- Larger number of unreliable nodes
- Enable by technology improvements in computing and networking
- Promise of P2P
  - Reliability – no central point of failure
    - Multiple replicas
    - Geographic distribution
  - High capacity through parallelism
  - Automatic configuration
  - ...

Distributed hash table (DHT)

- A simple interface – put(key, value), get(key) \(\rightarrow\) value
  - Goal - quick retrieval and storage of <key, value> pairs
- Supports a wide range of applications – no meaning assigned to keys

Application may be distributed over many nodes
DHT distributes data storage over many nodes
Common approach

- Assign random (160-bit) ID to each node
- Define a metric topology on the 160-bit numbers, i.e., the space of keys and node IDs
- Each node keeps contact info to $O(\log n)$
- Provide a lookup algorithm which finds the node whose ID is closest to a given key
  - Need a metric that identify closest node uniquely
- Store/retrieve a <key, value> pair at/from that node
Kademlia DHT

- Overarching idea
  - Treat nodes as leaves of a binary tree
  - Node position determined by shortest unique prefix of its ID
Kademlia DHT

- Overarching idea …
  - For any given node, divide the tree into a series of successively lower subtree that don’t contain the node
  - Protocol ensures every node knows at least of one node in each of its subtrees, if there is one
Kademlia DHT

- Overarching idea ...
  - To find a node of interest, a node successively query the best node it knows of to find a contact in lower and lower subtrees
  - Every hop brings it in a smaller subtree around the target
The XOR Metric

- Intuition – differences at higher order bits matter more than at lower order ones

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

distance is \(1 + 4 = 5\)

\textit{i.e., nodes in the same subtree are closer together than nodes in different subtrees}

- A valid metric
  - \(d(x,x) = 0\)
  - \(d(x,y) > 0\) if \(x \neq y\)
  - \(d(x,y) + d(y,z) \geq d(x, z)\)

- Also
  - XOR is unidirectional – for each \(x\) and \(t\), there is exactly one node \(y\) for which \(d(x,y) = t\)
  - XOR topology is also symmetric – \(d(x,y) = d(y,x)\)
Node state

- For each \( i \) \((0 \leq i < 160)\) every node keeps a list of \(<\text{IP-address, Port, Node id}>\) triples for nodes of distance between \(2^i\) and \(2^{(i+1)}\) from itself
  - Lists are called k-buckets

- Each k-bucket is sorted by time last seen, with most recently seen peer at the tail

- List can grow up to size \( k \), a system-wide replication parameter
  - Chosen so that any give set of \( k \) nodes is unlikely to fail within an hour (20)
Routing table data structure

For node with ID 00..0

K-buckets

3-buckets
Maintaining k-buckets

- Due to XOR topology’s symmetry, the distribution of nodes that call us is the same as that of contacts we need for our routing tables.
- The appropriate k-bucket (for the sender’s node id) is updated whenever a node receives a message:
  
  ```
  If sender is already in recipient’s k-bucket
    Move it to the tail
  Else
    If k-bucket is not full, insert at the tail
  Else
    Ping the least-recently seen node
    If it does not respond, evict it and insert sender at the tail
  Else discard sender
  ```

- A least-recently seen eviction policy, except that live nodes are never removed:
  - Based on distribution of peers’ lifespan
Joining, leaving and refreshes

- Node join
  - Borrow some contact from an already online node
  - Lookup itself
  - Cost of join is $O(\log n)$ messages
- Node leave – nothing to do
- Hourly k-bucket refreshes (if necessary)
Node lookup algorithm

- **Goal:** find the k nodes closest to a given target T
- **FIND_NODE**\(_n\): takes an ID as an argument, a recipient \(n\) returns \(<\text{IP address, UDP port, node id}>\) of the k nodes that it knows about closest to ID

**Lookup**

- \(n_0\) – ourselves (the node that is performing the lookup)
- \(N_1\) – find_node\(_{n_0}(T)\)
- \(N_2\) – find_node\(_{n_1}(T)\)
- …
- \(N_l\) – find_node\(_{n_{l-1}}(T)\)
  - This ends when \(N_l\) contains no contact that have not been called already

- \(n_i\) is any contact in \(N_i\)
Simple lookup
Concurrent lookup

- Trade bandwidth for lower latency lookups
- Goal
  - Route through closer/faster machines
  - Avoid delays due to timeout
- Idea – perform $\alpha > 1$ (3) calls to find_node$_n$(T) in parallel
- When $\alpha = 1$, lookup resembles that in Chord in terms of message cost, and the latency of detecting the failed nodes
- Unlike Chord, Kademlia has the flexibility of choosing any one of the k nodes in a bucket, so it can forward with lower latency
Kademlia protocol

- **PING**: to test whether a node is online
- **FIND_NODE**: takes an ID as an argument, a recipient returns `<IP address, UDP port, node id>` of the k nodes that it knows from the set of nodes closest to ID
- **FIND_VALUE**: behaves like FIND_NODE, unless the recipient received a STORE for that key, it just returns the stored value
- **STORE**: instruct a node to store a `<key, value>` pair for later retrieval
Store <key, value>

- Most operations implemented in terms of the lookup procedure
- To store a <key,value> pair, a participant locates the k closest nodes to the key and sends them STORE RPCs
- Each node re-publishes <key,value> pairs as necessary to keep them alive
- For Kademlia’s file sharing application, the original publisher of a <key,value> pair is required to republish it every 24 hours (otherwise it expires)
Find a `<key, value>`

- To find a `<key,value>` pair, perform a lookup of k closest nodes to the key
- Value lookup use FIND_VALUE, rather than FIND_NODE
- Procedure halts when any node returns the value
- Once a lookup succeed, requesting node stores `<key, value>` pair at the closes node observed that did not return the value
- Given unidirectionality of topology, future search will likely hit the cache
Kademlia today

- One of many DHTs out there: Chord, Pastry, Tapestry.
- The current de facto standard algorithm for P2P search
- It is used by LimeWire, Gnutella, Overnet, EDonkey2000, eMule, BitTorrent, among others
Conclusion

- **Operation cost**
  - As low as other popular protocols
  - Look up: $O(\log N)$, Join or leave: $O(\log 2N)$

- **Fault tolerance and concurrent change**
  - Handles well via the use of k-buckets

- **Proximity routing** -- chooses nodes that has low latency

- **Handles DoS attacks** by using that are up for a long time