Distributed Hashing

Aayush Kumar (CO21BTECH11002)
Vishal Vijay Devadiga (CS21BTECH11061)

What is Distributed Hashing?

Distributed Hashing is a technique used to distribute the keys of a hash table across multiple nodes in a network.

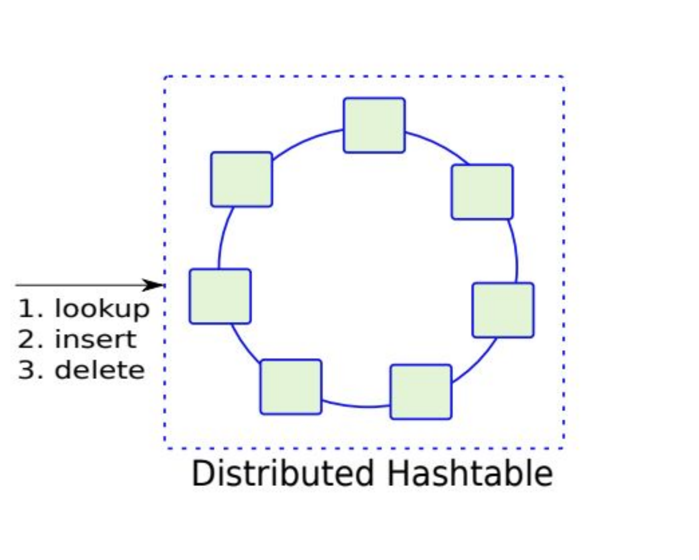
Which are the different types of Distributed Hash Tables?

There are different types of Distributed Hash Tables. Some of the most popular ones are:

1. Chord Distributed Hash Table (CDHT)
2. Kademlia Distributed Hash Table (KDHT)
3. Pastry Distributed Hash Table (PDHT)

Advantages

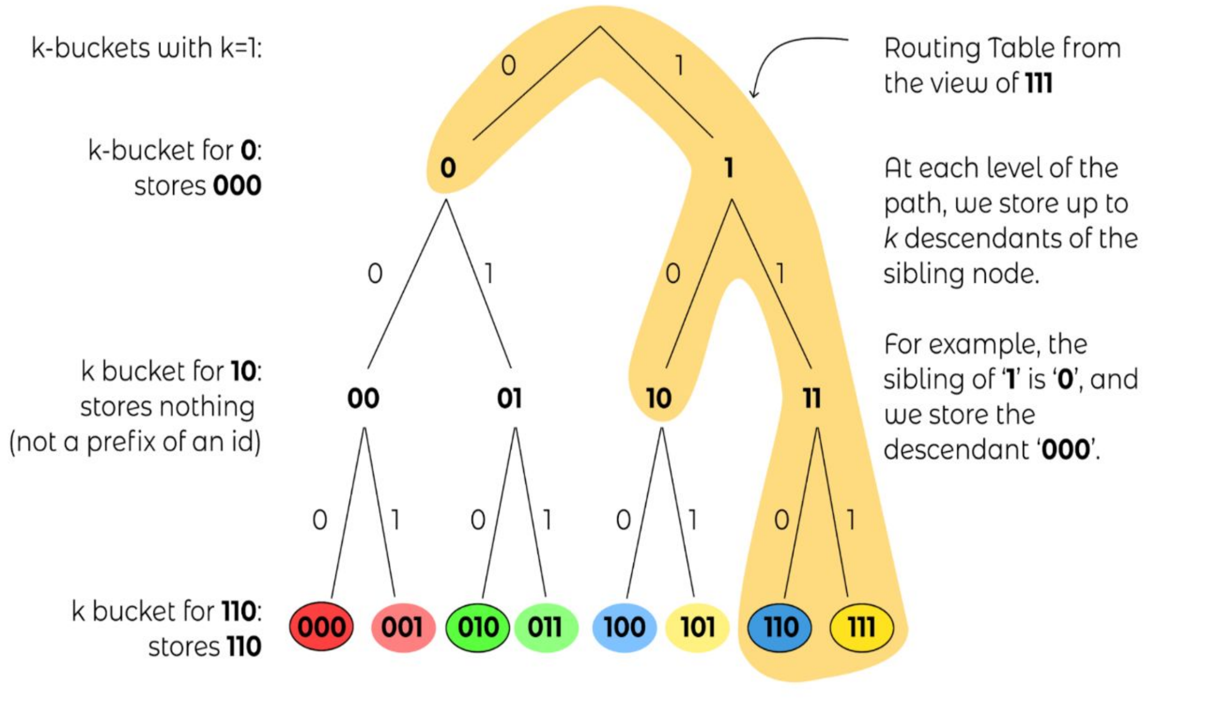
- DHTs scale, and are ideal candidates for web scale storage.
- They are more immune to node failures.
- DHTs also scale in terms of number of users. Different users are redirected to different nodes based on their keys. (better load balancing).



Focus of the Project: **KADEMLIA DHT**

- **NodeID** and **Key Space**: Kademlia represents nodes and keys with unique, fixed-length identifiers (NodeIDs and KeyIDs) in a binary space. The XOR distance between NodeIDs is used as a proximity metric.
- **Routing Table**: Each node maintains a routing table consisting of k-buckets, where each k-bucket corresponds to a different range of distances from the node. The k-buckets store the contact information of the k-closest nodes in that distance range.
- **Lookup Algorithm**: The process involves iterative queries to nodes progressively closer to the target KeyID. Each query returns a set of closer nodes, which are then used in subsequent queries.
- **Data Storage** and **Retrieval**: Data is stored as key-value pairs on the k closest nodes to the KeyID. A lookup is performed to retrieve data, and the value is returned from the first node that responds.

Focus of the Project: **KADEMLIA DHT**



k-buckets with k=1:

k-bucket for **0**:
stores **000**

k bucket for **10**:
stores nothing
(not a prefix of an id)

k bucket for **110**:
stores **110**

Routing Table from
the view of **111**

At each level of the path, we store up to *k* descendants of the sibling node.

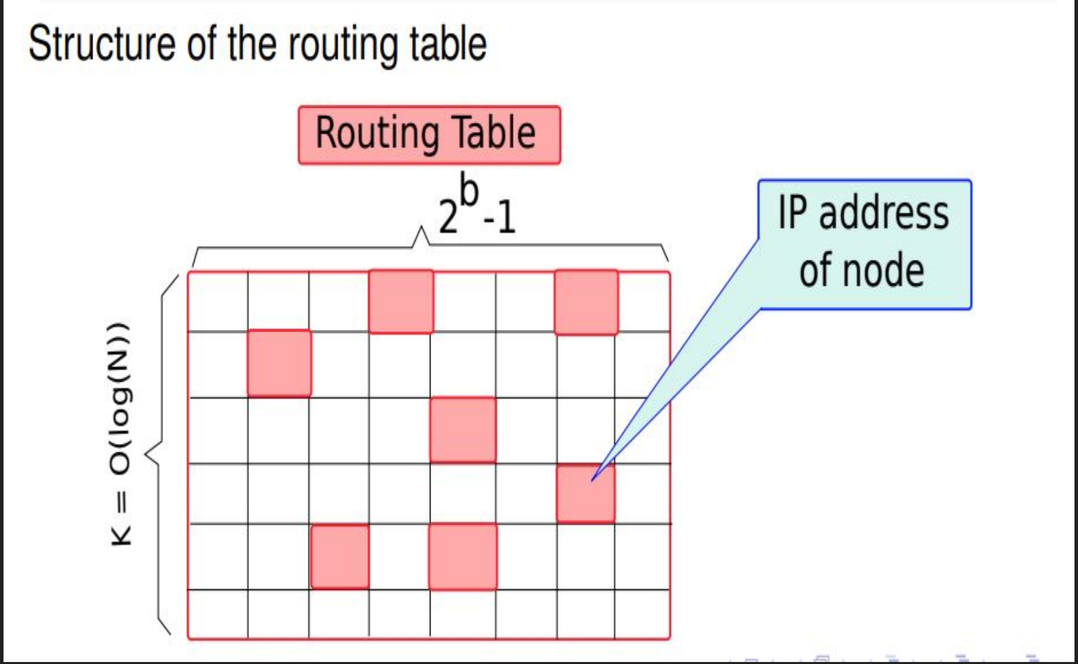
For example, the sibling of '**1**' is '**0**', and we store the descendant '**000**'.

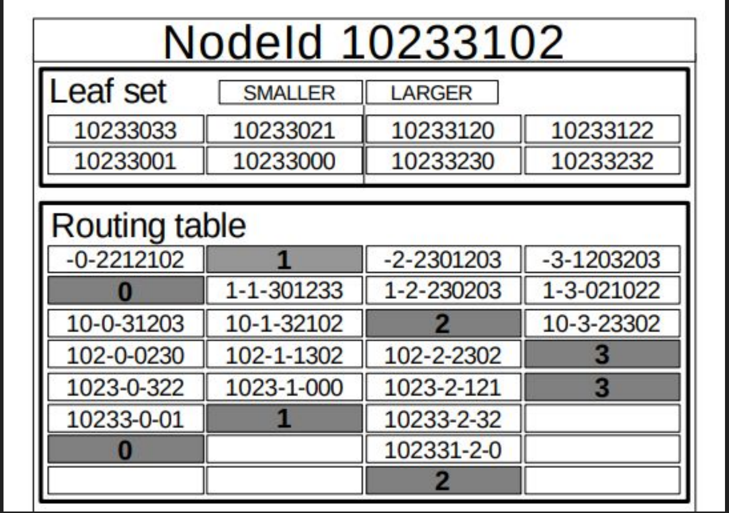
Figure 6: The routing table from the view of computer with id ***111****, with the corresponding* k*-buckets for k=1.*

Focus of the Project: **PASTRY DHT**

- **NodeID** and **Key Space**: Pastry assigns unique NodeIDs and KeyIDs in a circular space.
- **Routing Table**: Each node maintains a routing table with multiple levels representing a different prefix length. The table entries store the contact information of nodes with matching prefixes.
- **Leaf Set**: Besides the routing table, each node maintains a leaf set containing its closest neighbours in the circular space.
- **Lookup Algorithm**: The lookup process uses the routing table and leaf set to route messages towards nodes with increasingly longer prefix matches to the target KeyID.
- **Data Storage** and **Retrieval**: Data is stored in key-value pairs on the k closest nodes to the KeyID. Retrieval follows a similar lookup process as Kademlia and Chord.

Focus of the Project: **PASTRY DHT**





NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Deliverables

- Code: All code required to implement both DHTs: Pastry and Kademlia
- Application: Distributed Caches (DB)
  - We will be implementing a Distributed Cache that uses Hashing to store information
- Report Document containing:
  - Description of the Problem Statement
  - Details of the implementation of the DHTs
  - Comparison between the DHTs
  - Results