Distributed Hash Tables

Chord

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Comparison with Pastry

Chord vs Pastry

- Each node and each key's id is hashed to a unique value.
- The process of lookup tries to find the immediate successor to a key's id.
- The routing table at each node contains $O(log(n))$ entries.
- Inserting and deleting nodes requires $O(log(n)^2)$ messages.
- Sarangi View ☺ : More robust than Pastry, and more elegant.

Comparison with other Systems

- The Globe system assigns objects to locations, and is hieararchial. Chord is completely distributed and decentralized.
- CAN
  - Uses a d-dimensional co-ordinate space.
  - Each node maintains $O(d)$ state, and the lookup cost is $O(dN^{1/d})$.
  - Maintains a lesser amount of state than Chord, but has a higher lookup cost.

Features of Chord

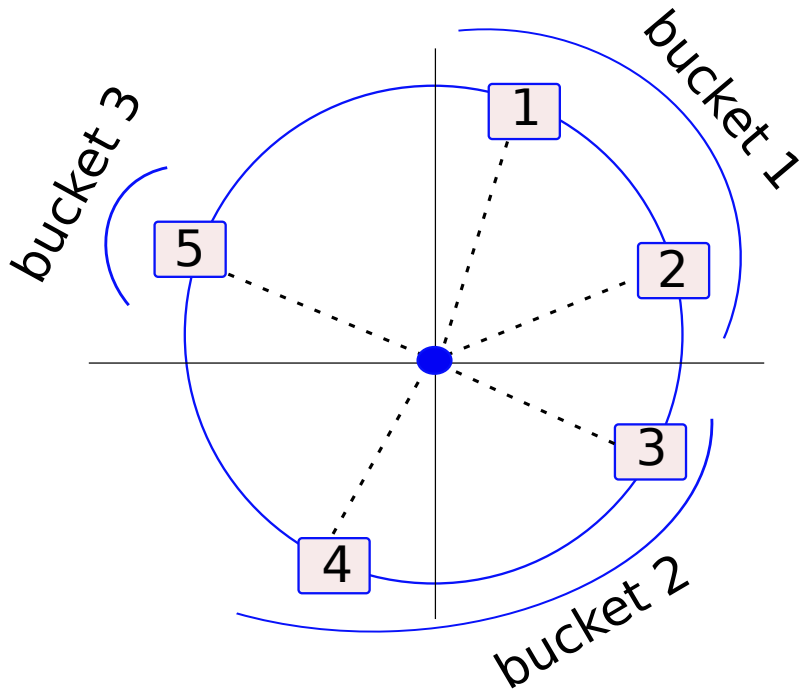
- Automatic load balancing
- Fully distributed
- Scalable in terms of state per node, bandwidth, and lookup time.
- Always available
- Provably correct.

Outline

Consistent Hashing

Definition

Consistent Hashing: It is a hashing technique that adapts very well to resizing of the hash table. Typically k/n elements need to be reshuffled across buckets. k is the number of keys and n is the number of slots in a hash table.



Structure of Chord

- Each node and key is assigned a m bit identifier.
- The hash for the node and key is generated by using the SHA-1 algorithm.
- The nodes are arranged in a circle (recall Pastry).
- Each key is assigned to the smallest node id that is larger than it. This node is known as the successor .

Objective

- For a given key, efficiently locate its successor.
- Efficiently manage addition and deletion of nodes.

Properties of Chord's Hashing Algorithm

- For n nodes, and k keys, with high probability
  1. Each node stores at most $(1+\epsilon)k/n$ keys
  2. Addition and deletion of nodes leads to a reshuffling of $O(k/n)$ keys
- Previous papers prove that $\epsilon = O(log(n))$
- There are techniques to reduce ϵ using virtual nodes.
  - Each node contains $log(n)$ virtual nodes.
  - Not scalable (Not necessarily required)

Chord's Routing(Finger) Table

Let m be the number of bits in an id

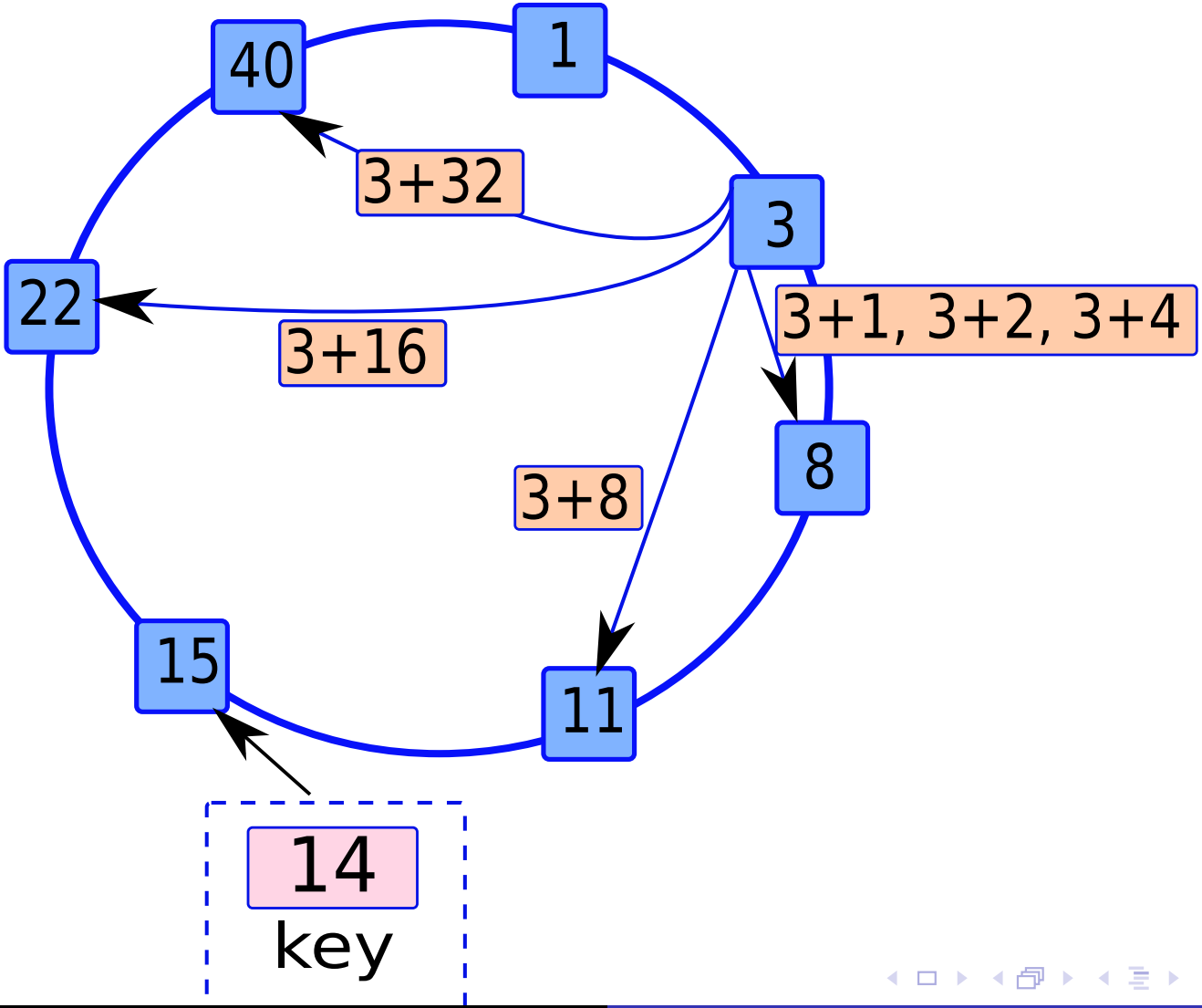
- Node n contains m entries in its finger table.
  - successor $\rightarrow$ next node on the identifier circle
  - predecessor $\rightarrow$ node on the identifier circle
- The i^{th} finger contains:
  - finger[i].start = (n + 2^{i-1}) mod 2^m, $(1 \leq i \leq m)$
  - finger[i].end = (n + $2^i - 1$) mod 2^m
  - finger[i].node = successor(finger[i].start)

Basic Operation

findSuccessor(keyId) $\rightarrow$ nodeId

Outline

Finger Table- II



Algorithms

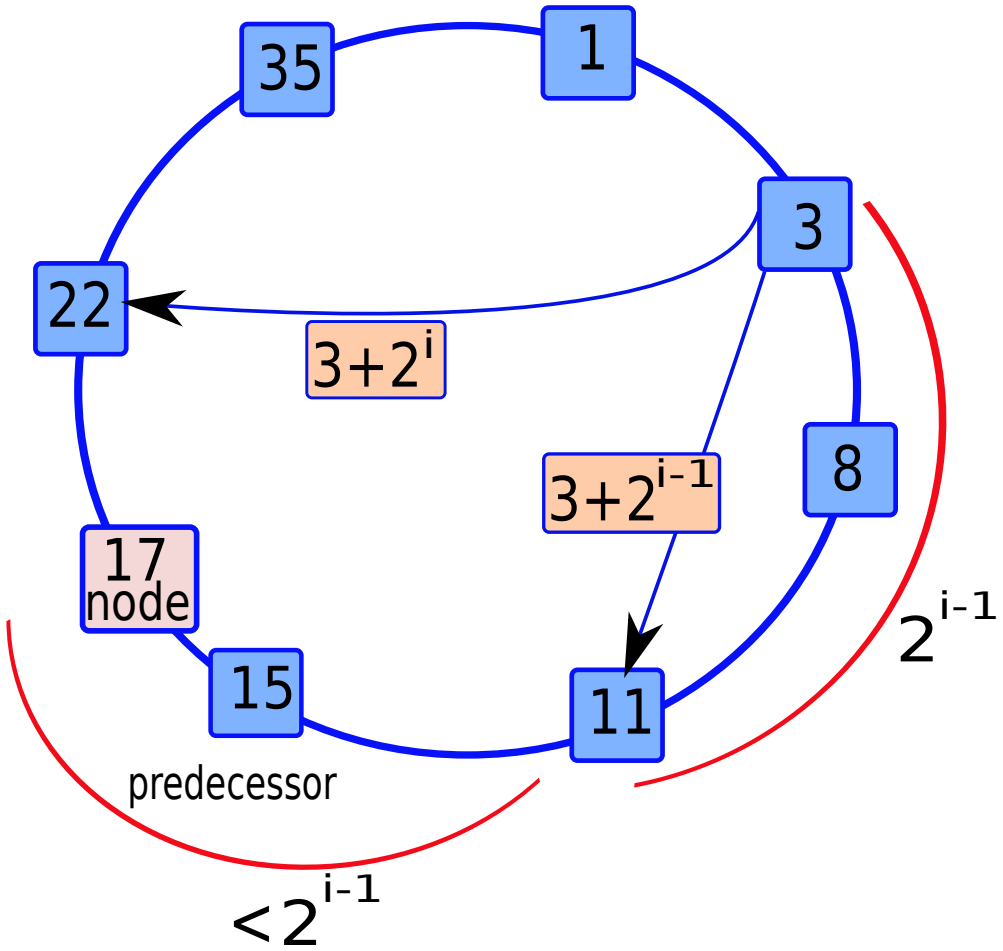
Algorithm 1: *findSuccessor* in Chord

```
1 n.findSuccessor(id) begin
2 |   n' ← findPredecessor(id)
  |   return n'.successor(id)
  |
3 end
4 n.findPredecessor(id) begin
5 |   n' ← n
  |   while id ∉ (n',n'.successor()) do
6 |   |   n' ← n'.closestPrecedingFinger(id)
7 |   end
8 |   return n'
9 end
```

closestPrecedingFinger(id)

```
1 n.closestPrecedingFinger(id) begin
2     for i ← m to 1 do
3         if finger[i].node ∈ (n, id) then
4             return finger[i].node
5         end
6     end
7     return n
8 end
```

$O(log(n))$ Routing Complexity



Outline

Node Arrival

Each node maintains a predecessor pointer

- Initialize the predecessor and the fingers of the new node.
- Update the predecessor and fingers of other nodes
- Notify software that the node is ready

Node Arrival - II

n initially contacts n'

```
1 n.join(n') begin
2 |   n.initFingerTable(n')
  |   updateOthers()
3 end
```

Algorithm 2: *initFingerTable* in Chord

```
1  n.initFingerTable(n') begin
2      finger[1].node ← n'.findSuccessor(finger[1].start)
       successor ← finger[1].node
       predecessor ← successor.predecessor
       successor.predecessor ← n
       predecessor.successor ← n
       for i ← 1 to m-1 do
3          if finger[i+1].start ∈ (n, finger[i].node) then
4              finger[i+1].node ← finger[i].node
5          end
6          else
7              finger[i+1].node ← n'.findSuccessor(finger[i+1].start)
8          end
9      end
```

updateOthers()

```
1  n.updateOthers() begin
2      for i ← 1 to m do
3          pred ← findPredecessor (n - 2^(i-1))
           pred.updateFingerTable(n, i)
4      end
5  end
6  pred.updateFingerTable(n, i) begin
7      if n ∈ (pred, finger[i].node) then
8          finger[i].node ← n
           p ← predecessor
           p.updateFingerTable(n,i)
9      end
10 end
```

Stabilization of the Network (run periodically)

```
1  n.stabilize() begin
2      x ← successor.predecessor
       if x ∈ (n, successor) then
3          successor ← x
4      end
5      successor.notify(n)
6  end
7  n.notify(n') begin
8      if (predecessor is null) OR (n' ∈ (predecessor, n)) then
9          predecessor ← n'
10     end
11 end
```

Stabilization-II

```
1 n.fix_fingers() begin
2 |   i ← random()
  |   finger[i].node ← find_successor (finger[i].start)
3 end
```

Results

Evaluation Setup

- Network consists 10^4 nodes
- Number of keys : 10^5 to 10^6
- Each experiment is repeated 20 times
- The major results are on a Chord protocol simulator

Effect of Virtual Nodes

- The number of keys per node decreases with the number of virtual nodes .
- For 1 virtual node, we can have up to 500 keys per node (mean 100).
- For 10 virtual nodes, we can have roughly 50 to 200 keys per node (mean 100).

source [1]

Average Path Length

- The path length in Chord grows with the number of nodes.
- It is roughly normally distributed about the mean. For a mean of 6 nodes (path length), the $\pm 3\sigma$ range varies from 1 to 11.

Number of nodes	Path length (approx.)
10	2
100	3
1000	4.3
10000	6.2

source [1]

Other DHT Systems: Tapestry

Tapestry

- 160-bit block id, Octal digits
- Routing table like Pastry (digit based hypercube)
- Does not have a leaf set or neighborhood table.

Other DHT Systems: Kademlia

Kademlia

- Basis of BitTorrent
- Each node has a 128-bit id
- Each digit contains only 1 bit
- Find the closest node to a key
- Values are stored at several nodes
- Nodes can cache the values of popular keys.

Other DHT Systems: CAN

CAN – Content Addressable Network

- It uses a d-dimensional multi-torus as its overlay network.
- Node uses standard routing algorithms for tori. It uses $O(d)$ space. (Note: This is independent of n)
- Each node contains a virtual co-ordinate zone.
- Node Arrival: Split a zone
- Node Departure: Merge a zone

Chord: A Peer-to-Peer Lookup Service for Internet Applications, by I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Proc. ACM SIGCOMM, San Diego, CA, September 2001.