Distributed Hash Tables II

Dominic Duggan
Stevens Institute of Technology

FINGER TABLES

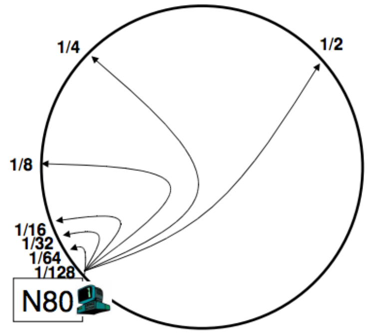
2

Finger Tables

M = #nodes

m = #key bits

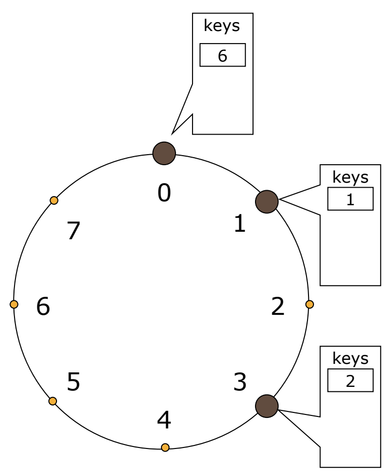
$M = 2^m$



- Entry *i* in the finger table of node N is the first node that succeeds or equals $ID_N + 2^i$ (mod M)
- $FINGER_N(i) = \min \{ID_{N2} \mid ID_{N2} \geq ID_N + 2^i \pmod{M}\}$
- i.e. i^{th} finger points $1/2^{m-i}$ way around the ring

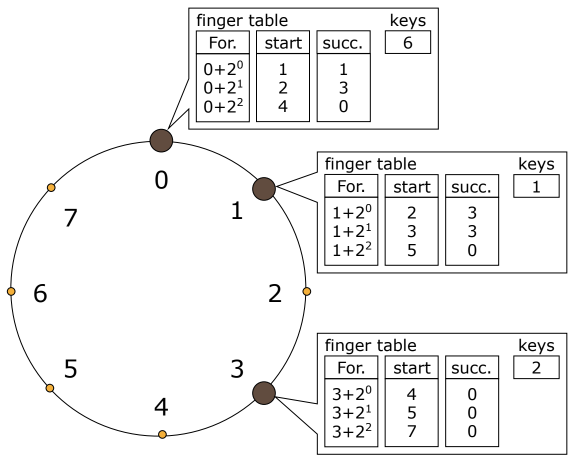
3

Example Ring



4

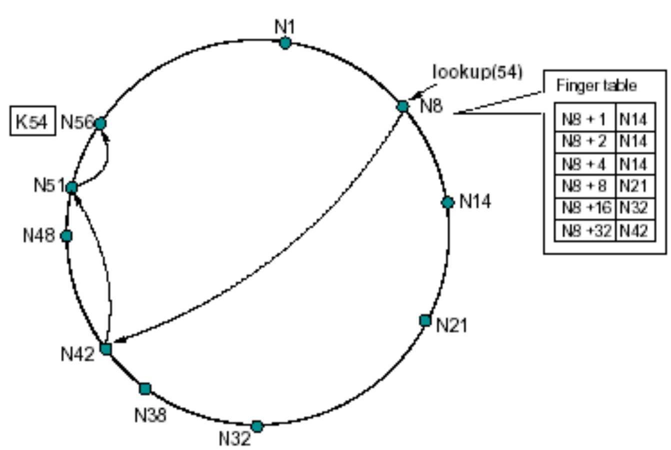
Finger Tables



5

Example query

- The path of a query for key 54 starting at node 8:



6

Node Join and Stabilization

- "Stabilization" protocol contains 6 functions:
  - `create()`
  - `join()`
  - `stabilize()`
  - `notify()`
  - `fix_fingers()`
  - `check_predecessor()`

Node Join – `join()`

- When node n first starts, it calls `n.join(n')`, for some node `n'`
- `join()` function asks `n'` to find the immediate successor of n

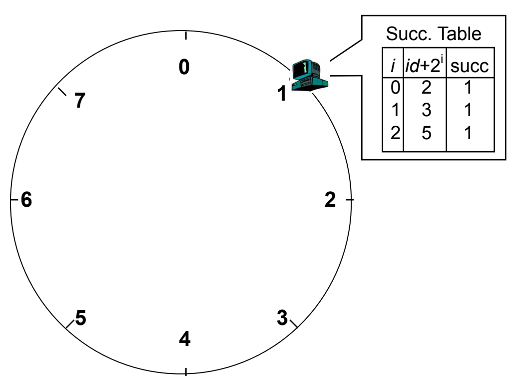
Node Join – join()

```
// create a new Chord ring.
n.create()
  predecessor = nil;
  successor = n;

// join a Chord ring containing node n’.
n.join(n’)
  predecessor = nil;
  successor = n’.find_successor(n);
```

9

DHT: Chord Join

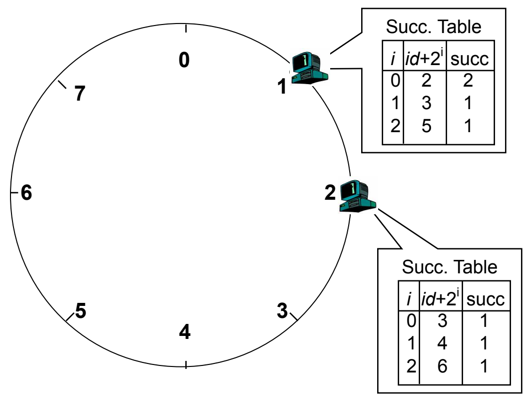
- Assume an identifier space [0..7]
- Node n1 joins



10

DHT: Chord Join

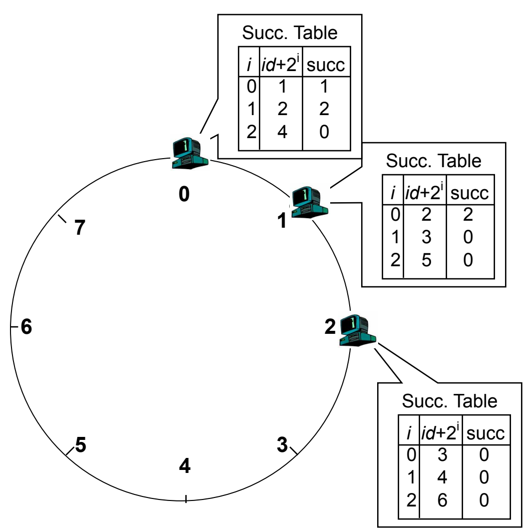
- Node n2 joins



11

DHT: Chord Join

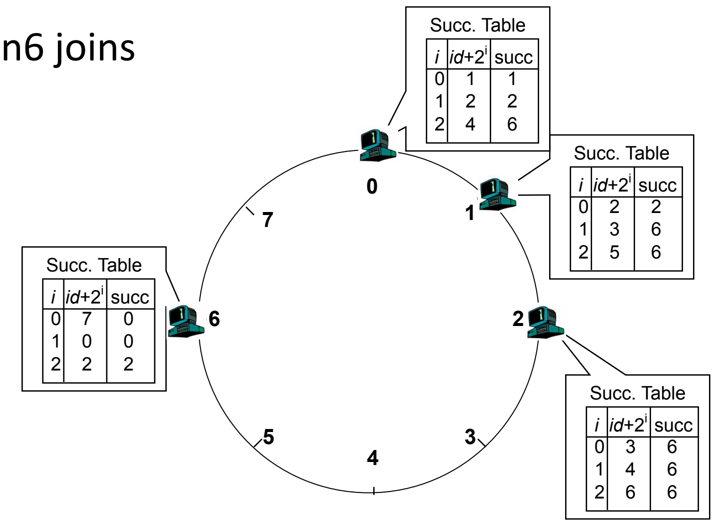
- Node n0 joins



12

DHT: Chord Join

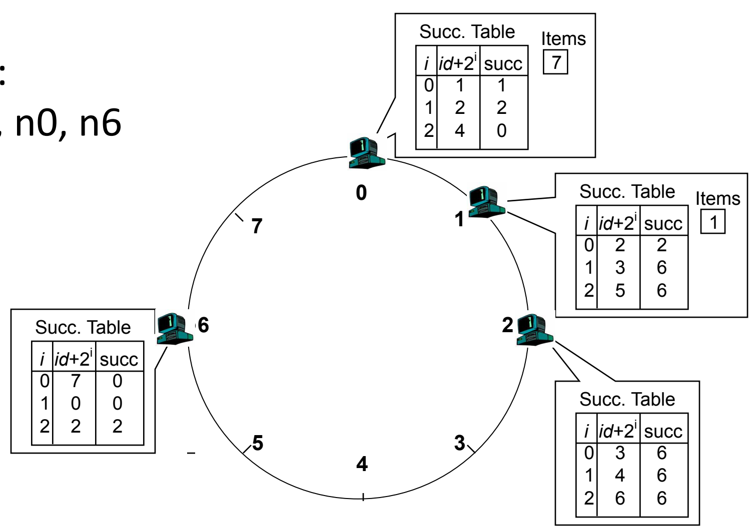
- Node n6 joins



13

DHT: Chord Join

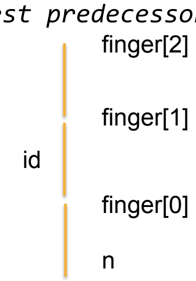
- Nodes: n1, n2, n0, n6
- Items: f7, f1



14

Scalable Key Location – find_successor()

```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
      return successor;
  else
      n’ = closest_preceding_node(id);
      return n’.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m-1 downto 0
      if (finger[i] ∈ (n, id))
            return finger[i];
  return n;
```



15

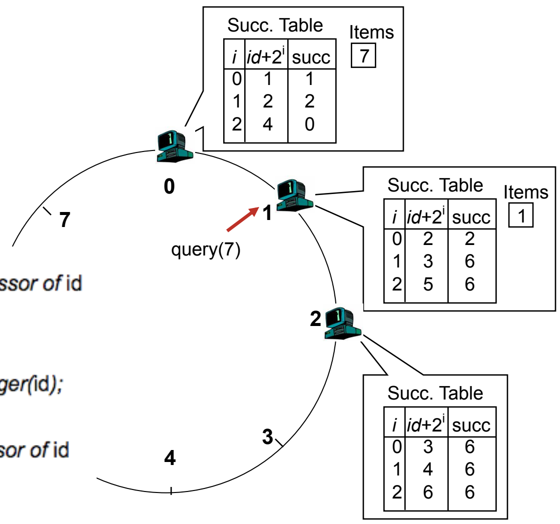
Scalable Key Location – find_successor()

```
// ask node n to find the successor of id
n.find_predecessor(id):
  n’ = n
  while (id ∉ (n’, n’.successor])
      n’ = n’.closest_preceding_finger(id)
  return n’

n.find_successor(id):
  n’ =  find_predecessor(id)
  return n’.successor

// search the local table for the highest predecessor of id
n.closest_preceding_finger(id)
  for i = m-1 downto 0
      if (finger[i] ∈ (n, id))
            return finger[i];
  return n;
```

16

DHT: Chord Routing

- Upon receiving a query for item id, a node:
  - Checks whether stores the item locally
  - If not, forwards the query to the largest node in its successor table that does not exceed id

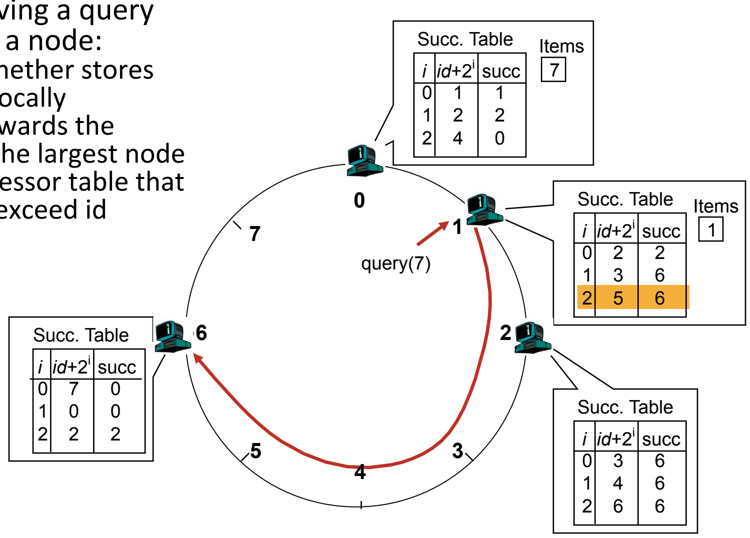



```
// ask node n to find the predecessor of id
n.find_predecessor(id)
  n' = n;
  while (id ∉ (n', n'.successor])
    n' = n'.closest_preceeding_finger(id);
  return n'

// ask node n to find the successor of id
n.find_successor(id)
  n' = find_predecessor(id);
  return n'.successor;
```

17

DHT: Chord Routing

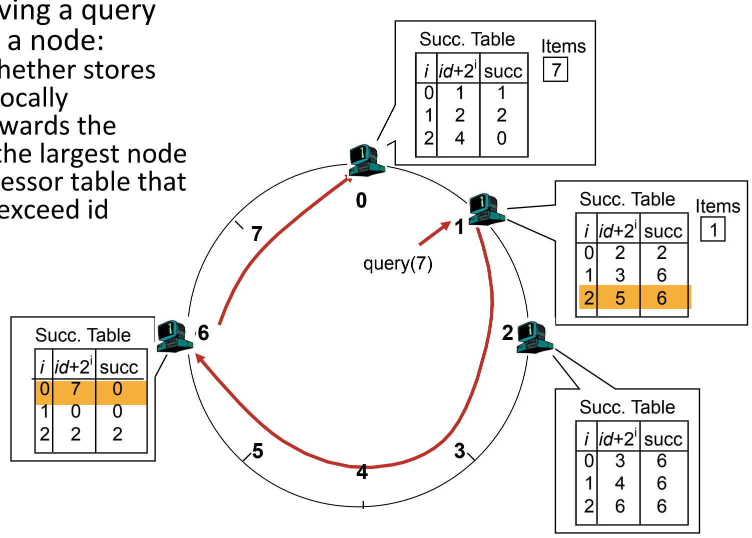
- Upon receiving a query for item id, a node:
  - Checks whether stores the item locally
  - If not, forwards the query to the largest node in its successor table that does not exceed id



18

DHT: Chord Routing

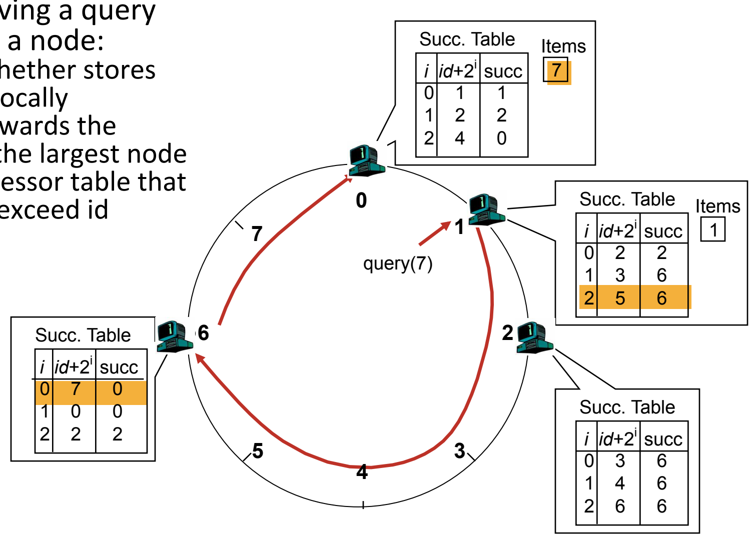
- Upon receiving a query for item id, a node:
  - Checks whether stores the item locally
  - If not, forwards the query to the largest node in its successor table that does not exceed id



19

DHT: Chord Routing

- Upon receiving a query for item id, a node:
  - Checks whether stores the item locally
  - If not, forwards the query to the largest node in its successor table that does not exceed id



20

Chord reliability

- Correct routing table (successors, predecessors, and fingers)
- Primary invariant: correctness of successor pointers
  - Fingers for performance
  - Algorithm is to "get closer" to the target
  - Successor nodes always do this

Join: Relaxed Approach

- If ring is correct, then routing is correct
- Stabilization
  - Each node periodically runs stabilization routine

Join: Relaxed Approach

- If ring is correct, then routing is correct
  - Fingers needed for speed only
- Stabilization
  - Each node periodically runs stabilization routine
  - Each node refreshes all fingers by periodically calling `find_successor(N+2^(i-1))` ($N+2^{i-1}$) for random `i`
  - Periodic cost is O(log M) per node due to finger refresh

23

fix_fingers()

```
// called periodically. refreshes finger table entries.
n.fix_fingers()
  next = next + 1 ;
  if (next > m-1)
      next = 0;
  finger[next] = find_successor(n + 2^(next-1));

// checks whether predecessor has failed.
n.check_predecessor()
  if (predecessor has failed)
      predecessor = nil;
```

24

DHT: Chord Summary

- Pros:
  - Guaranteed Lookup
  - O(log M) per node state and search scope
- Cons:
  - Supporting non-exact match search is hard

25