TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Distributed hash table (DHT)

Lecturer: Thanh-Chung Dao
Slides by Viet-Trung Tran
School of Information and Communication Technology

1

Outline

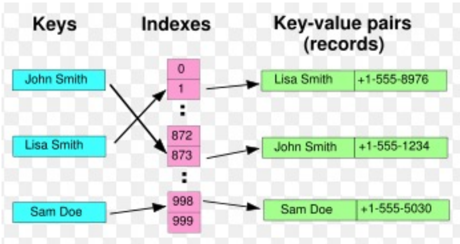
- Hashing
- Distributed Hash Table
- Chord

2

2

A Hash Table (hash map)

- A data structure implements an associative array that can map keys to values.
  - searching and insertions are 0(1) in the worse case
- Uses a hash function to compute an index into an array of buckets or slots from which the correct value can be found.
  - index = f(key, array_size)



3

3

Hash functions

- Crucial for good hash table performance
- Can be difficult to achieve
  - WANTED: uniform distribution of hash values
  - A non-uniform distribution increases the number of collisions and the cost of resolving them

4

4

Hashing for partitioning usecase

- Objective
  - Given document X, choose one of k servers to use
- Eg. using modulo hashing
  - Number servers 1..k
  - Place X on server i = (X mod k)
    - Problem? Data may not be uniformly distributed
  - Place X on server i = hash (X) mod k
- Problem?
  - What happens if a server fails or joins (k → k±1)?
  - What is different clients has different estimate of k?
  - Answer: All entries get remapped to new nodes!

5

5

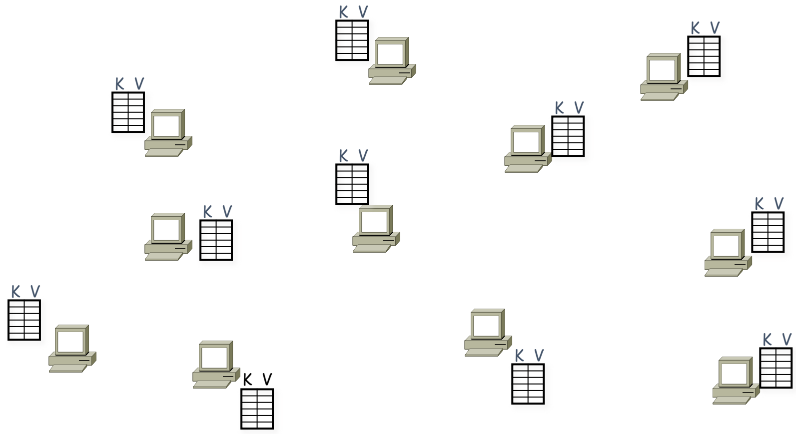
Distributed hash table (DHT)

- Distributed Hash Table (DHT) is similar to hash table but spread across many hosts
- Interface
  - insert(key, value)
  - lookup(key)
- Every DHT node supports a single operation:
  - Given key as input; route messages to node holding key

6

6

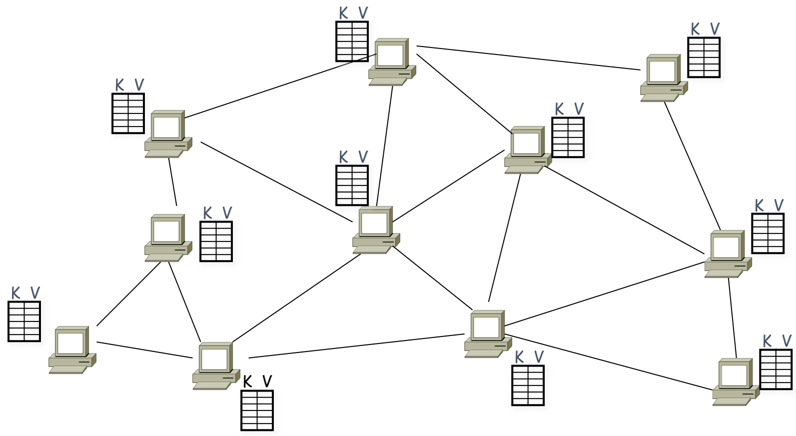
DHT: basic idea



7

7

DHT: basic idea

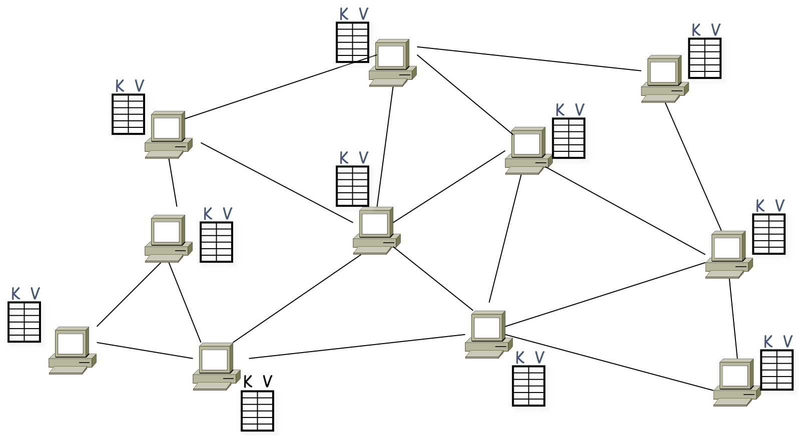


Neighboring nodes are “connected” at the application-level

8

8

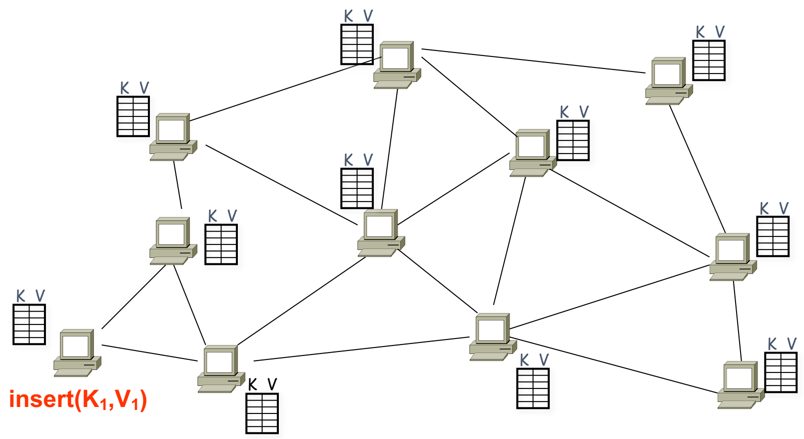
DHT: basic idea



Operation: take *key* as input; route messages to node holding *key* 9

9

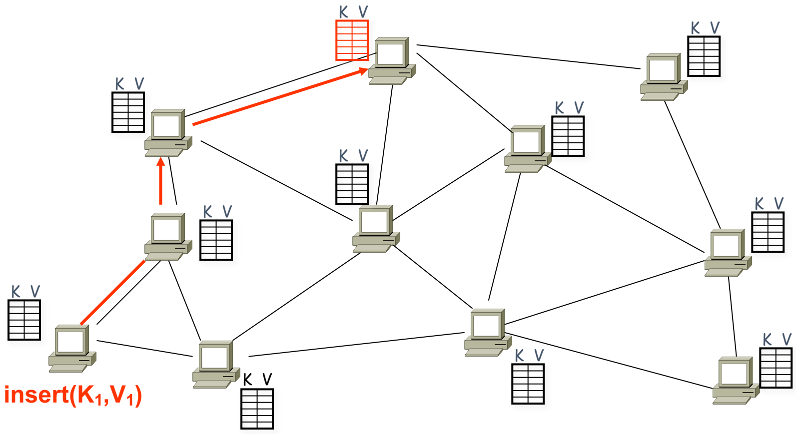
DHT: basic idea



Operation: take *key* as input; route messages to node holding *key* 10

10

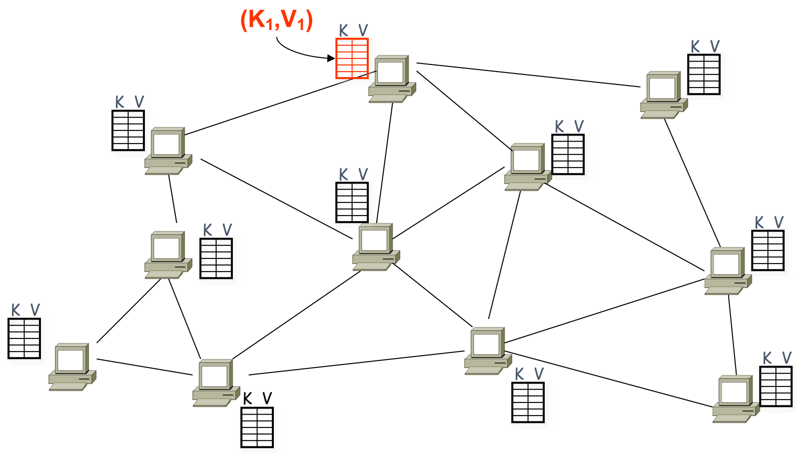
DHT: basic idea



Operation: take *key* as input; route messages to node holding *key* 11

11

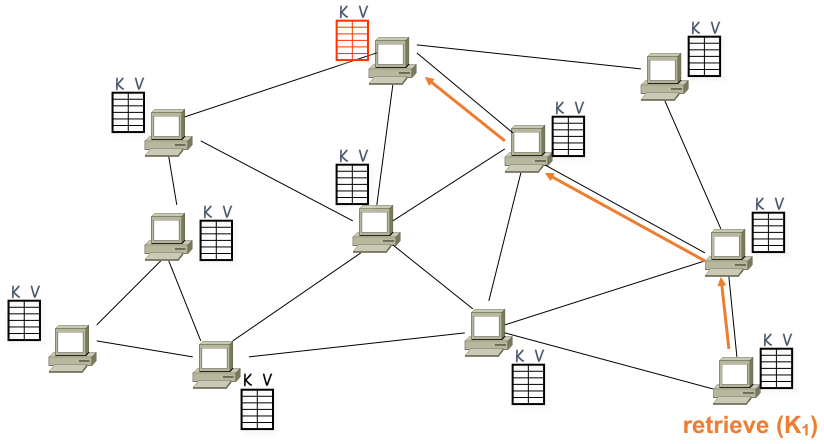
DHT: basic idea



Operation: take *key* as input; route messages to node holding *key* 12

12

DHT: basic idea



Operation: take *key* as input; route messages to node holding *key*

13

13

How to design a DHT?

- State Assignment
  - What "(key, value) tables" does a node store?
- Network Topology
  - How does a node select its neighbors?
- Routing Algorithm:
  - Which neighbor to pick while routing to a destination?
- Various DHT algorithms make different choices
  - CAN, Chord, Pastry, Tapestry, Plaxton, Viceroy, Kademlia, Skipnet, Symphony, Koorde, Apocrypha, Land, ORDI …

14

14

Chord: A scalable peer-to-peer look-up protocol for internet applications

Credit: University of California, berkely and Max planck institute

15

15

Outline

- What is Chord?
- Consistent Hashing
- A Simple Key Lookup Algorithm
- Scalable Key Lookup Algorithm
- Node Joins and Stabilization
- Node Failures

16

16

What is Chord?

- In short: a peer-to-peer lookup system
- Given a key (data item), it maps the key onto a node (peer).
- Uses consistent hashing to assign keys to nodes .
- Solves the problem of locating key in a collection of distributed nodes.
- Maintains routing information with frequent node arrivals and departures

17

Consistent hashing

- Consistent hash function assigns each node and key an m-bit identifier.
- SHA-1 is used as a base hash function.
- A node's identifier is defined by hashing the node's IP address.
- A key identifier is produced by hashing the key (chord doesn't define this. Depends on the application).
  - ID(node) = hash(IP, Port)
  - ID(key) = hash(key)

18

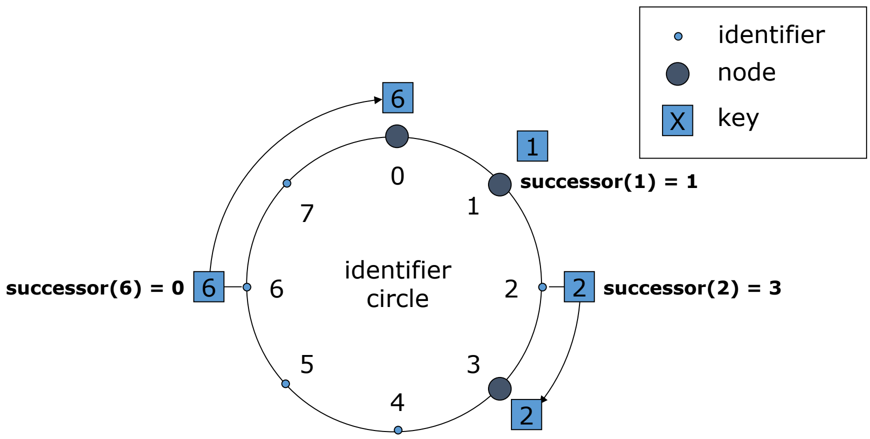
Consistent hashing

- In an m-bit identifier space, there are 2^m identifiers.
- Identifiers are ordered on an identifier circle modulo 2^m.
- The identifier ring is called Chord ring.
- Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space.
- This node is the successor node of key k, denoted by successor(k).

19

19

Consistent hashing – Successor nodes



20

20

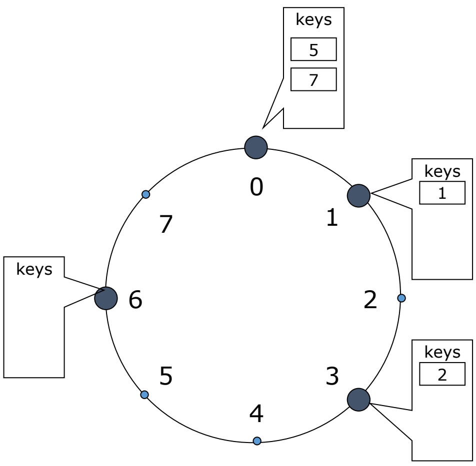
Consistent hashing – Join and departure

- When a node n joins the network, certain keys previously assigned to n’s successor now become assigned to n.
- When node n leaves the network, all of its assigned keys are reassigned to n’s successor.

21

21

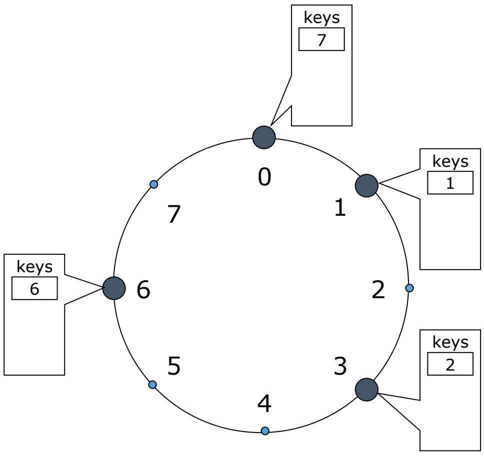
Consistent hashing – Node join



22

22

Consistent hashing – Node departure



A Simple key lookup

- If each node knows only how to contact its current successor node on the identifier circle, all node can be visited in linear order.
- Queries for a given identifier could be passed around the circle via these successor pointers until they encounter the node that contains the key.

A Simple key lookup

- Pseudo code for finding successor:

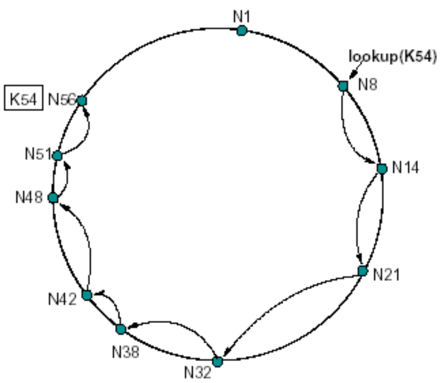
```
// ask node n to find the successor of id
n.find_successor(id)
    if (id ∈ (n, successor])
        return successor;
    else
        // forward the query around the circle
        return successor.find_successor(id);
```

25

25

A Simple key lookup

- The path taken by a query from node 8 for key 54:



26

26

Scalable key location

- To accelerate lookups, Chord maintains additional routing information.
- This additional information is not essential for correctness, which is achieved as long as each node knows its correct successor.

27

27

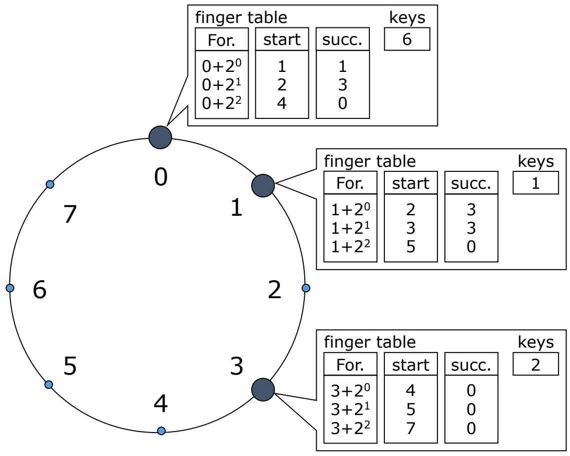
Scalable key location – Finger tables

- Each node n' maintains a routing table with up to m entries (which is in fact the number of bits in identifiers), called finger table.
- The ith entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^i-1 on the identifier circle.
- s = successor(n+2^i-1).
- s is called the ith finger of node n, denoted by n.finger(i)

28

28

Scalable key location – Finger tables



29

29

Scalable key location – Finger tables

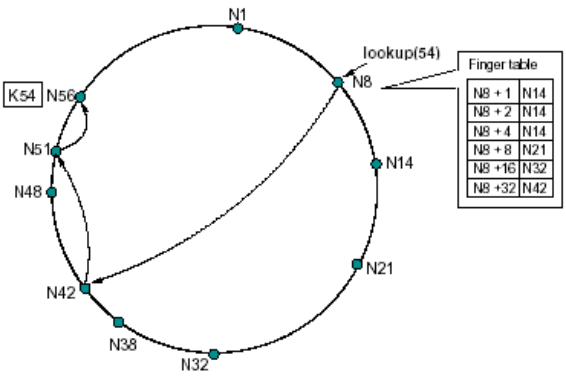
- A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node.
- The first finger of n is the immediate successor of n on the circle.

30

30

Scalable key location – Example query

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



31

31

Applications: Chord-based DNS

- DNS provides a lookup service
  - keys: host names values: IP adresses
- Chord could hash each host name to a key
- Chord-based DNS:
  - no special root servers
  - no manual management of routing information
  - no naming structure
  - can find objects not tied to particular machines

46

46

What is Chord? – Addressed problems

- **Load balance**: chord acts as a distributed hash function, spreading keys evenly over nodes
- **Decentralization**: chord is fully distributed, no node is more important than any other, improves robustness
- **Scalability**: logarithmic growth of lookup costs with the number of nodes in the network, even very large systems are feasible
- **Availability**: chord automatically adjusts its internal tables to ensure that the node responsible for a key can always be found
- **Flexible naming**: chord places no constraints on the structure of the keys it looks up.

47

47

Summary

- Simple, powerful protocol
- Only operation: map a key to the responsible node
- Each node maintains information about O(log N) other nodes
- Lookups via O(log N) messages
- Scales well with number of nodes
- Continues to function correctly despite even major changes of the system

48

48

Thanks for your attention!

49

49