Name Resolution in Flat Name Spaces

Distributed Hash Tables (DHTs)

Pedro F. Souto (pfs@fe.up.pt)

April 3, 2019

Resolution of Unstructured Names

Problem

- Assume you want to develop a "peer-to-peer" version of the backup service on the Internet.
- How do you locate the peers storing a given chunk of a file?
  - Each file has a 256-bit id
  - This id is **unstructured**

No solution Broadcasting/multicasting

- It just does not scale beyond a LAN

Issue How do we **resolve** efficiently an unstructured name on the Internet?

Solution Use a distributed hash table (DHT)

- Answer provided by academia to the problem of locating an entity in P2P system

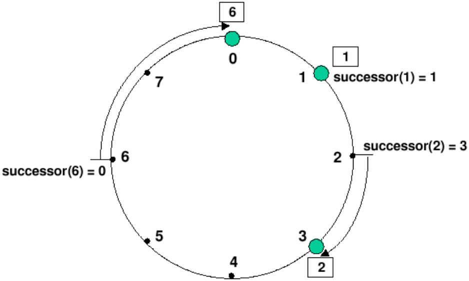
Distributed Hash Table (DHT)

- A DHT is similar to a **hash-table**
  - It maps a **key** to **value**
  - The **key** is an object identifier
  - The **value** is an address
    - assume it is the address of the node/peer **responsible** for the key
- A DHT provides a single operation:
  lookup(key) returns the address of the node responsible for the key
  - The address can be used to insert an object, to access to an object ...
- In a DHT-based system, node identifiers and key values are drawn from the same domain, e.g. a number with m bits
- The node responsible for a key value is the one whose identifier is **closer** to that key
  - Depending on the definition of **distance** we get different DHTs

DHT Example: Chord

- Chord uses identifiers with m-bits ordered in a ring ($mod 2^m$)
- Each "object" has an m-bit random identifier: the key of DHT entries ($m = 128$ in the original paper - used MD5)
  - Obtained by hashing the object's key
- Each node has an m-bit random identifier
  - Obtained, e.g., by hashing the node's IP address
- The node **responsible** for key k is the **successor** of key k, $succ(k)$:

  $succ(k)$ is the node with the **smallest** id that is larger or equal to k ($succ(k) \geq k$, in modular arithmetic)
  - Given a key k the node responsible for it will have an id **higher or equal** to k.



src: Stoica et. al. 2001

Key Resolution in Chord (1/2)

Problem Given a key k, how do you find $succ(k)$?

No Solution 1 Each node n keeps information about its **successor**, i.e. the next node in the ring ($succ(n+1)$)

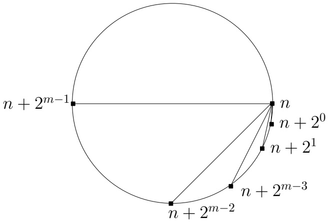
- Can use any resolution strategy (iterative, transitive or recursive)
- ... but it does not scale. Why?

No Solution 2 Each node n keeps information about all nodes in the ring

- Constant time name resolution
- ... but it does not scale. Why?

Key Resolution in Chord (2/2)

Solution In addition to a pointer to the next node in the ring each node keeps pointers that allow it to reduce at least in half the **distance** to the key



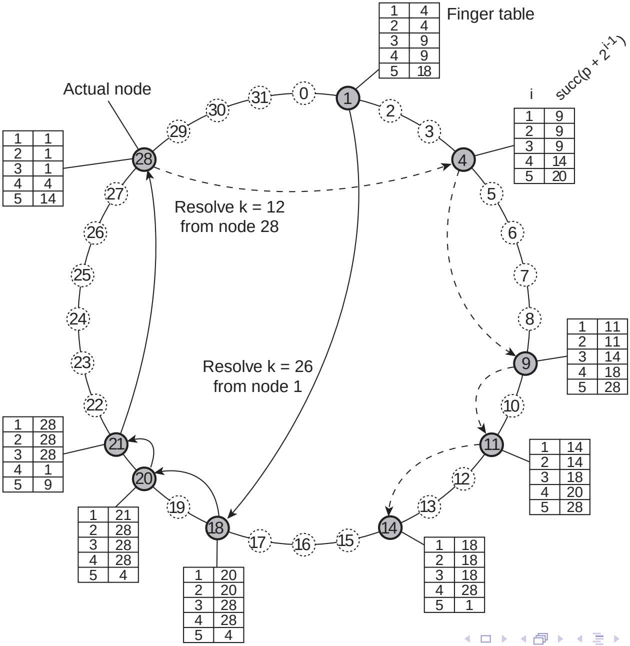
- Because nodes that are 2^i apart may not be active, each node n keeps a pointer to the $succ(n + 2^i)$ for $i = 0 \ldots m - 1$

- This scheme has 3 important properties:
  1. Each node keeps information on only m nodes
  2. Each node knows more about nodes closer to it than nodes farer away
  3. The table in a node may not have information on the $succ(k)$, for some k – i.e. a node may be unable to resolve a key by itself
- Key resolution requires $O(log(N))$ steps, where N is the number of nodes in the system

Chord: *Finger Table* (1/2)

- The ***Finger table***, $FT_n[]$, is an array with m pointers:
  $FT_n[i] = succ(n + 2^{i-1}) mod 2^m$ where $i = 1 \ldots m$
  - $FT_n[1]$ is n's successor in the Chord ring
- To resolve (*lookup*) a key k, node n forwards the request to:
  - The next node, i.e. $FT_n[1]$, if $n < k \leq FT_n[1]$
  - To node $n' = FT_n[j]$, where j is s.t. $FT_n[j] < k < FT_n[j+1]$
    (All arithmetic in modulo 2^m)
- Algorithmically, n' can be computed by:
  1. Traversing the FT from the last to the first element
  2. Stopping at the element $FT_n[j]$ st: $n < FT_n[j] < k$
- Each element of the FT includes not only the node identifier but also its IP address (and port)
- Chord works correctly iff $FT_n[1]$ is correct
  - Chord tolerates transient inconsistencies in other elements of $FT_n[]$, by trying the resolution again (may not be necessary even)

Chord: *Finger Table* (2/2)



Chord: Other Issues

Node Joining Node n can ask any node to locate $succ(n)$

- The crux is to get the $FT_x[1]$ correct
- Every node needs also to keep information about its **predecessor**
- Periodically:
    1. A node queries its successor about its predecessor, and notifies its successor, after updating its successor, if needed
    2. Updates the elements of its FT, one at a time
    3. Checks if its predecessor is still in the ring

Node Failure Rather than keep a single successor, a node keeps a list of r successors

- If the successor fails, a node can replace it with next one from that list

Identifiers Generation To achieve some tolerance to denial-of-service (DoS) attacks, identifiers should be generated using a cryptographic hash function, e.g. SHA256

Virtual Topology Issues (1/2)

Problem Chord, and other P2P systems, use an overlay network

- If the topology of the overlay network is oblivious to the underlying physical network, routing of messages along the overlay network may be inefficient
  - Messages may follow an erratic route, e.g. bouncing between hosts in different continents

Sol. 1: Assign identifiers according to the underlying topology

- I.e. assign identifiers so that the overlay topology is close to that of the underlying physical topology.
- This is not always possible. E.g. it is **not** possible in Chord.

Virtual Topology Issues (2/2)

Sol. 2: Route messages according to the underlying topology

- For example, Chord could keep several nodes per interval $[n+2^{i-1}, n+2^i]$ rather than a single one, and when resolving a key, might use the closest node

Sol. 3: Pick neighbors according to the underlying topology

- In some algorithms, nodes can pick their neighbors, i.e. establish the links of the overlay network.
- This is not always possible. E.g. it is **not** possible in Chord.

Further Reading

- Subsection 5.2.3, Tanenbaum and van Steen, *Distributed Systems*, 2nd Ed.
- I. Stoica et al., "Chord: A scalable peer-to-peer lookup protocol for Internet applications", IEEE/ACM Transactions on Networks, (11)1:17-32, Feb 2003 (acessível via biblioteca digital da ACM “dentro da FEUP”)