Supplement: Distributed hash table and Chord

Sep 13, 2021

Prof. Shinnazar Seytnazarov

Faculty of Computer Science & Engineering

Recap - Structured P2P

❑ Essence

- Overlay network is constructed using a deterministic topology:
  - ring, binary tree, grid, etc
- Make use of a semantic-free index: each data item is uniquely associated with a *key*, in turn used as an *index*. Common practice: use a hash function

  key(data item) = hash(data item's value)
- P2P system now responsible for storing (*key,value*) pairs.

Recap - Structured P2P

- **Distributed hash table (DHT) is the most used mechanism**
  - Data items are assigned a random key from a large identifier space
  - Nodes are assigned a random number from the same space
  - Efficient and deterministic scheme uniquely mapping the key of a data item to the identifier of a node using some distance metric
  - When looking up a data item, the network address of the node responsible for that data item is returned
  - Many DHT variations (e.g. Chord, CAN, Pastry, Bamboo, Tapestry, Kademlia)

Consistent hashing

- **Say we want to store information about books on 4 nodes.**
  - Use the ISBN to identify each book.
  - We could use one of the nodes as a central directory server
- **But, with the hash of the ISBN, we don't need a central server:**

```
switch (SHA-1(ISBN) mod 4) {
        case 0: // store on node1
        case 1: // store on node2
        case 2: // store on node3
        case 3: // store on node4
}
```

- **Our store gets bigger.....we need to add more 2 nodes.**
  - We now must recalculate where all the books are stored.
- **Do the books stay on the same nodes?**
  - The only books stored on the same node as before are those where SHA-1(ISBN) mod 4 == SHA-1(ISBN) mod 6

Consistent hashing

❑ What it gives

- Consistent hashing allows you to add more nodes and only a small minority of books will have to move to new nodes.

❑ Key property

- Low cost hashtable expansion.
  - That is, a book's hash key is independent of the number of books and independent of the number of nodes.
  - If you add or remove nodes or books, a book's hash key remains the same.
- Mechanism: hash something constant at each node
  - E.g., a node's MAC address

Example: Chord

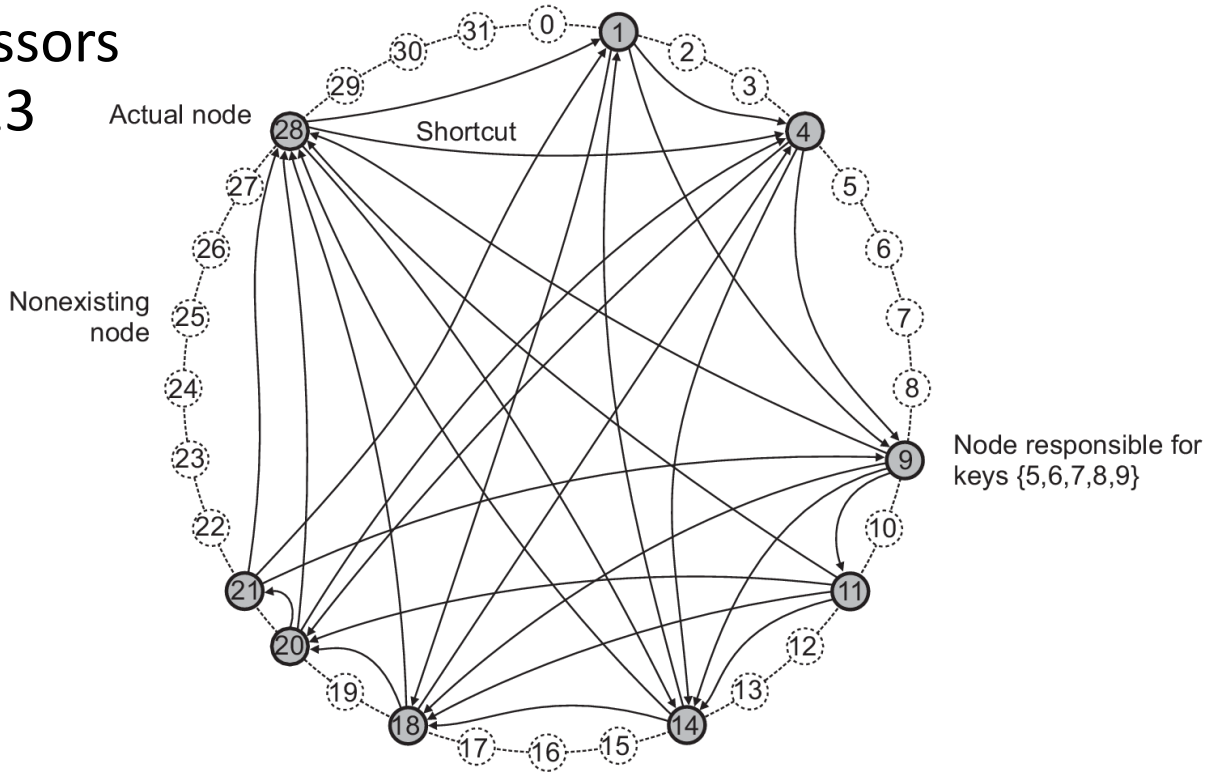
❑ Principle

- Nodes are logically organized in a ring.
- Each node has an *m*-bit identifier.
- Each data item is hashed to an *m*-bit key.
- Data item with key *k* is stored at node with smallest identifier *id* ≥ *k,* called the successor of key *k* and denoted as *succ(k)*
- The ring is extended with various shortcut links to other nodes.

Example: Chord

❑ Chord

- m = 5-bit identifier
- Only nine nodes
- They are successors for remaining 23



Example: Chord

❑ Locating the address efficiently

- The main issue in DHT-based systems is to efficiently resolve a key *k* to the address of *succ(k)*

Bad solution

- Let each node *p* keep track of the successor *succ(p + 1)* as well as its predecessor *pred(p)*
- whenever a node p receives a request to resolve key *k*, it will simply forward the request to one of its two neighbors
  - whichever one is appropriate
  - unless $pred(p) < k \leq p$ in which case node *p* should return its own address to the process that initiated the resolution of key *k*.

Better solution

- Finger table!

Example: Chord

❑ Finger table

Principle

- Each Chord node maintains a finger table containing $s \leq m$ entries. If FT_p denotes the finger table of node p, then

$$FT_p[i] = succ\left(p + 2^{i-1}\right) \text{ where } i \in [1, m]$$

- Let's build FT for node $p = 1$ and for $s = 5$ entries
  - Node has only three outgoing connections: 4, 9, and 18

i	$succ\left(p + 2^{i-1}\right)$
1	4
2	4
3	9
4	9
5	18

Example: Chord

❑ Lookup

Principle

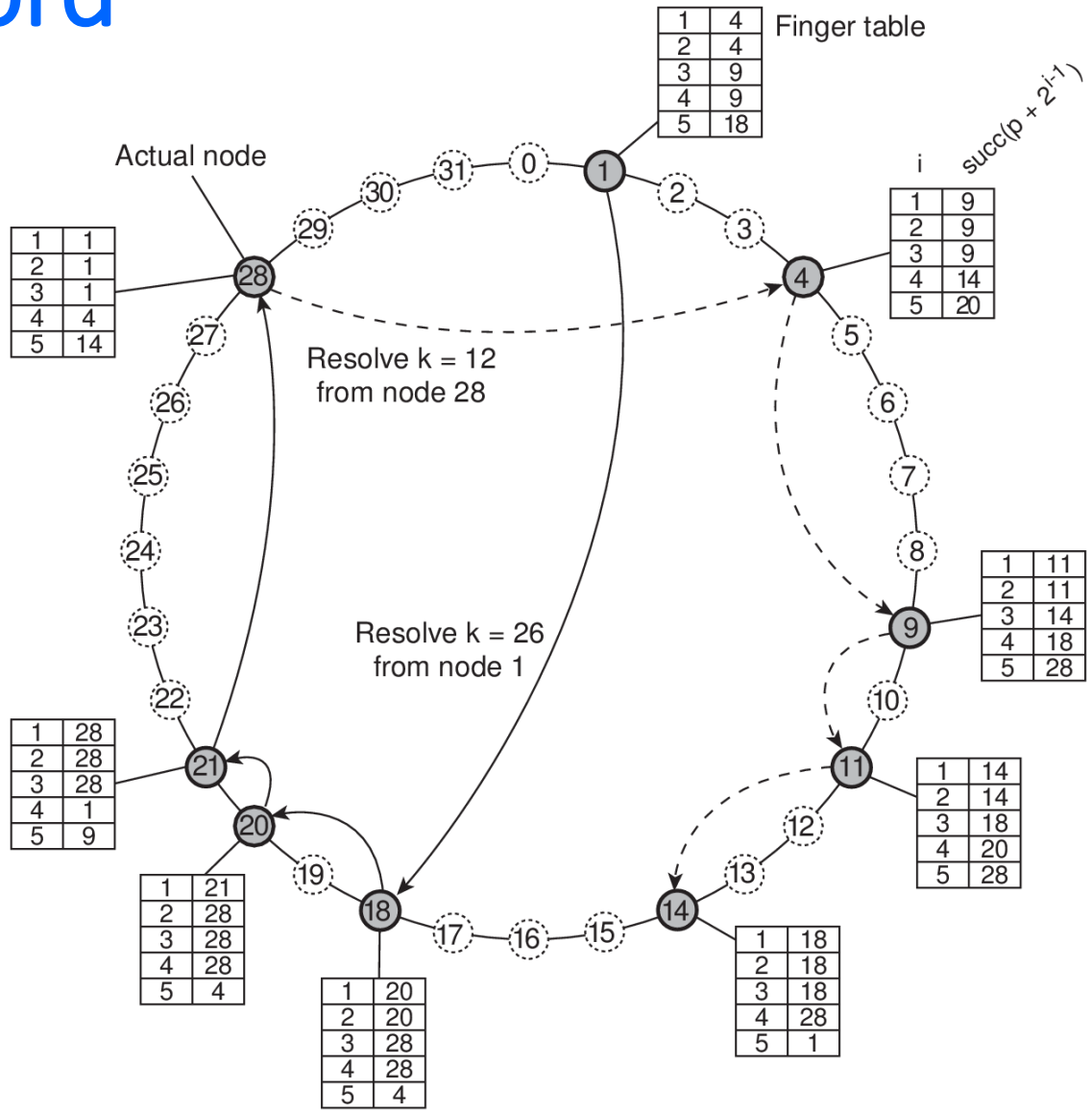
- To look up a key *k*, node *p* forwards the request to node with index *j* in finger table satisfying

$$q = FT_p[j] \leq k < FT_p[j+1]$$

- In other words, node *p* will try to forward the request "as far as possible" but without passing it beyond the node responsible for that key

Example: Chord

- **Examples:**
  - Lookup(26)@1
  - Lookup(12)@28



Any questions?