CS 555: Distributed Systems [Chord]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

Topics covered in this lecture

- Chord
  - Mapping of data items
  - Data lookups
  - Finger Table construction
  - Assimilation of peers

Distributed hash tables: Identifiers

- Data items are assigned an identifier from a large random space
  - 128-bit UUIDs or 160-bit SHA1 digests
- **Nodes are also assigned a number from the same identifier space**

Crux of the DHT problem

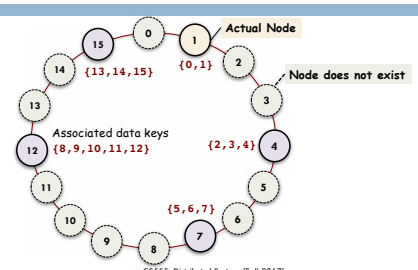
- Implement an efficient, **deterministic** scheme to
  - Map data items to node
- When you **look up** a data item
  - Network address of node holding the data is returned

CHORD

The Chord System

- Assigns IDs to keys and nodes from the same 1-dimensional ID space
- Nodes are organized into a **ring**
- Data item with key **k** is mapped to a node with the **smallest** **id ≥ k**
  - Also referred to as successor(**k**)

Mapping of data items to nodes in Chord



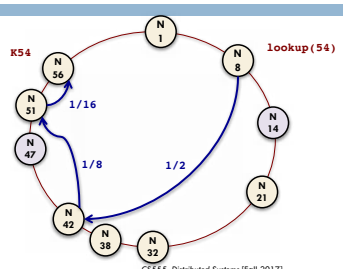
Chord lookups

- **N** is the number of possible nodes in the system
- Each node maintains a **finger table**
  - With log N entries
  - Entries contains IP addresses of nodes
    - Half-way around the ID space from it
    - $1/4^{th}$, $1/8^{th}$, … **in powers of two**
  - Ensures node can forward lookup query to at least ½ of the remaining ID-space distance to key
    - Lookups in *O(log N)*

Storing keys and forwarding lookups

- An entity with key *k* falls under the ***jurisdiction*** of node with the *smallest identifier id*
  - *id >= k*
  - Referred to as the successor of *k* or *succ(k)*
- A node ***forwards*** query for key *k* to node (in its FT) with highest ID < *k*

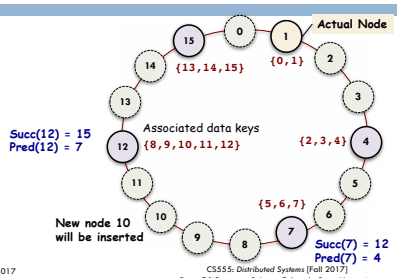
Chord lookup example for k=54



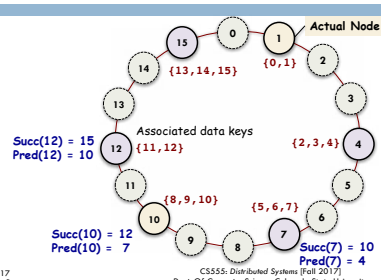
When a node wants to join

- Generate a random id
  - Probability of collisions is low
- **lookup(id)**
  - Will return successor(id)
- Contact *successor(id)* and its predecessor
  - Insert self in the ring
  - **Transfer** data items
    - All keys must be fetched from the new node's successor

An example of inserting a new node



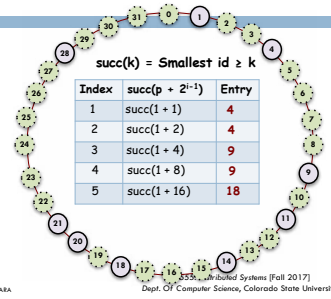
An example of inserting a new node



Finger Table in Chord

- Chord uses an m-bit identifier space
  - 2^m possible peers
- Each node, p, in Chord maintains a Finger Table with m-entries
  - $FT_p[i] = succ(p + 2^{i-1})$

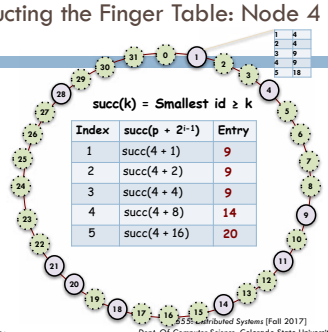
Constructing the Finger Table: Node 1



succ(k) = Smallest id ≥ k

Index	succ(p + 2^{i-1})	Entry
1	succ(1 + 1)	4
2	succ(1 + 2)	4
3	succ(1 + 4)	9
4	succ(1 + 8)	9
5	succ(1 + 16)	18

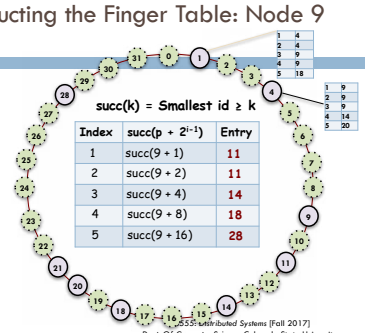
Constructing the Finger Table: Node 4



succ(k) = Smallest id ≥ k

Index	succ(p + 2^{i-1})	Entry
1	succ(4 + 1)	9
2	succ(4 + 2)	9
3	succ(4 + 4)	9
4	succ(4 + 8)	14
5	succ(4 + 16)	20

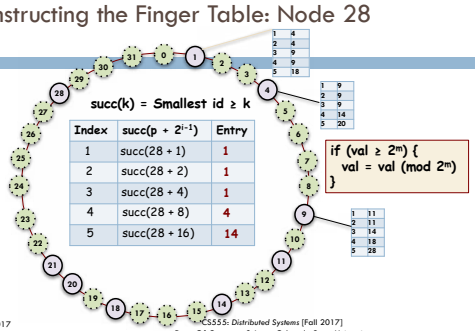
Constructing the Finger Table: Node 9



succ(k) = Smallest id ≥ k

Index	succ(p + 2^{i-1})	Entry
1	succ(9 + 1)	11
2	succ(9 + 2)	11
3	succ(9 + 4)	14
4	succ(9 + 8)	18
5	succ(9 + 16)	28

Constructing the Finger Table: Node 28



Index	succ(p + 2^{i-1})	Entry
1	succ(28 + 1)	1
2	succ(28 + 2)	1
3	succ(28 + 4)	1
4	succ(28 + 8)	4
5	succ(28 + 16)	14

Using the finger table to route queries: Make sure you don't overshoot

- To lookup a key k, node p will forward query to node q with index j in p's FT where:

Node with greatest ID less than k

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

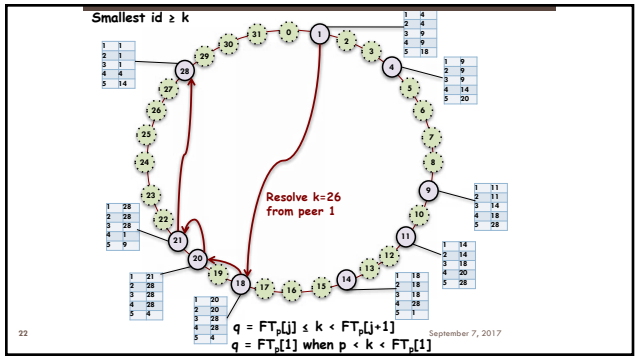
OR

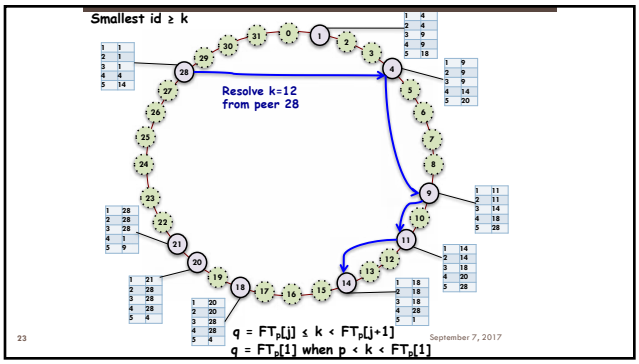
$$q = FT_p[1] \text{ when } p < k < FT_p[1]$$

First entry ONLY if its ID is greater than k

Stop forwarding the query when you are the target node

- A node is ***responsible*** for keys that fall in the range
  - *key > predecessor*
  - *key <= self*



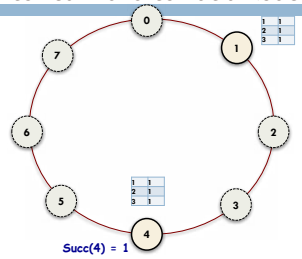


Keeping the finger table up-to-date: At node q, $FT_q[1]$ must be accurate

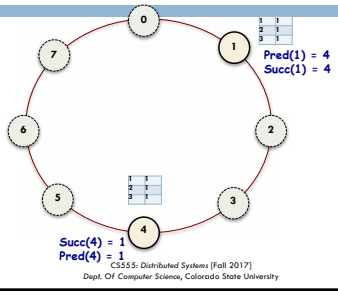
1. Contact *succ(q+1)* {This is $FT_q[1]$}
   - Have it return its predecessor
2. If $q = pred(succ(q+1))$
   - Everything is fine
3. Otherwise:
   - There is a new node p such that $q < p \le succ(q+1)$
   - $FT_q[1] = p$
   - Check if p has recorded q as its predecessor
     No? Go to step (1)

AN EXAMPLE OF NODES JOINING IN CHORD

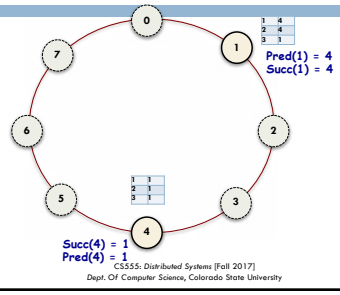
An example of inserting a new node N-4: Node-4 comes in and contacts Node-1



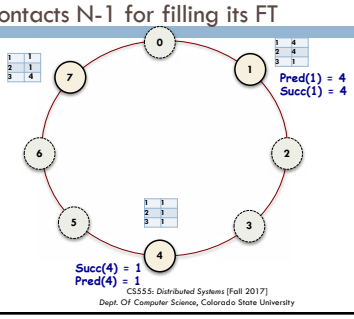
Installing successor at Node-1



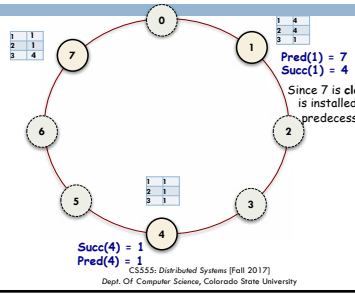
Updating the FT at N-1



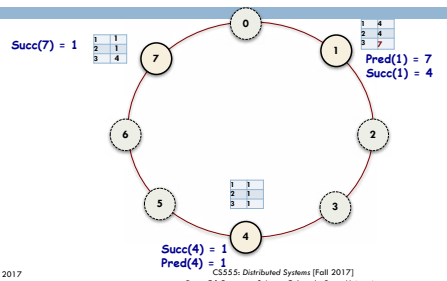
An example of inserting a new node N-7: N-7 contacts N-1 for filling its FT



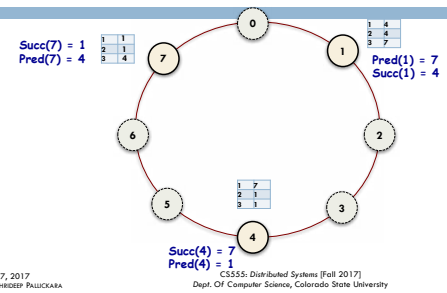
N-7 informs N-1 that it (N-7) is now N-1's predecessor



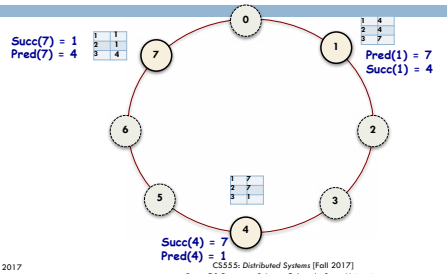
When N-1 updates its FT later on …



N-4 contacts N-1 to see if it is still its predecessor … and installs N-7 as its successor



When the FT at N-4 is updated …



Pastry

Pastry

- All nodes and objects are assigned 128-bit GUIDs
- Applies secure hash function to:
  - The public-key assigned to each node ➔ Node GUID
  - The object's name or some part of the object's stored state

Resulting GUIDs have usual properties of secure hash values

- They are **randomly distributed** in the range $0 - (2^{128} - 1)$
- Provide no clue about the values from which they were computed
- **Collisions** in the GUID space (for nodes and objects) are *extremely unlikely*

The Pastry routing

- The number of nodes in the network, N
- The algorithm will correctly route messages addressed to any GUID in O(log N) steps
  - Delivered to an active node whose GUID is numerically closest to it
- **Active nodes** take responsibility for processing requests addressed to all objects in their *numerical neighborhood*

Pastry routing

- Routing transfers message to a node that is **closer** to its destination
- Closeness is in an artificial space
  - The space of GUIDs

Minimizing unnecessarily extended transport paths

- Pastry uses a **locality metric** based on network distance
  - Hop-counts, round-trip delay measurements
- Uses locality metric to select appropriate neighbors when setting up the routing tables

Managing **churn**: Nodes joining and leaving the system

- Fully self-organizing
- When new nodes join the overlay?
  - Obtain data needed to construct routing table and other required state from existing members
    - In O(log N) messages: **N** is the number of hosts in overlay
- When a node fails or departs?
  - Remaining nodes detect its absence
  - Nodes **cooperatively reconfigure** to reflect required changes in routing structure
    - In O(log N) messages

The contents of this slide-set are based on the following references

- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van der Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273.* [Chapter 5]
- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 10]