Tapestry & DHT Wrapup

Arvind Krishnamurthy
Fall 2003

Chord/CAN Summary

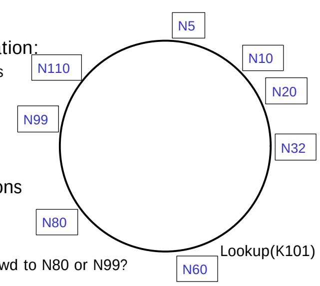
- Each node "owns" some portion of the key-space
  - In Chord, it is the key-id-space between two nodes in 1-D ring
  - In CAN, it is a multi-dimensional "zone"
- Routing is implicit
  - Node X does not know of a path to a key Z
  - But it knows that Node Y will have better information to get to Z
  - So route to Y
- Files and nodes are assigned random locations in key-space
  - Provides load balance
    - Probabilistically equal division of keys to nodes
  - Question?
    - What other notions of load balance are satisfied/not-satisfied?

Tricky Issues

- Node join:
  - Need to find a bootstrap node
  - No aesthetically nice solutions for bootstrapping
- Node fails:
  - Need to transport keys to "adjacent" node
  - What other issues arise from node failure?

Routing Optimizations

- Random distributions destroy locality
  - Each overlay hop could potentially travel half the diameter of the internet
- Chord locality optimization:
  - Do cost-benefit analysis
  - Cost: latency cost
  - Benefit: distance covered in key-id space
- What other optimizations can we perform?



CAN optimizations

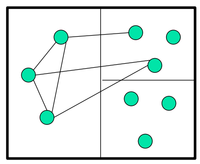
- Routing distance: $(dn^{1/d})/4$
  - Let's say n=10000, d=2, Avg. distance = 50
  - Let's say n=10000, d=4, Avg. distance = 10
- Increasing "d":
  - Lowers hop-count
  - Increases local state (avg. neighbors: 2d)
- CAN locality optimization #1:
  - Many possible routes from each node to some target node
  - Choose which dimension to route through first
  - Again perform cost-benefit analysis
    - Metric is (latency/distance in key-id space)

CAN: Multiple realities

- Use different hash functions and maintain a CAN for each hash function
  - Each node has "2*d" neighbors in each reality
- Replicate keys
  - Gives fault tolerance
  - Can route to any one of the different copies in the system
  - Choose a target based on the closeness in the id-space

CAN: multiple nodes per zone

- Each zone is occupied by multiple nodes
- Nodes is aware of:
  - All nodes in the zone
  - Picks closest neighbor from its neighboring zone



CAN: Distributed Binning

- Avoid randomization of nodes to zones
- Use landmark beacons to identify zones
  - Divide coordinate space into k! regions using k beacons
  - Based on relative ordering of the closest beacons
- Effects:
  - Improves locality
  - Destroys load balance
  - Highly dependent on the choice of beacons

DHT general discussion

- DHTs provide a simple interface:
  - Insert(key, info)
  - Lookup(key) → info
- DHTs have been used to build file systems
- Is DHT the right abstraction? Should we replace kazaa by DHT-like systems for file sharing?

Announcements

- Assignment 2 design due today
  - Signup for design review meetings on Monday/Tuesday
- First of two quizes:
  - Scheduled for October 13th (Monday)

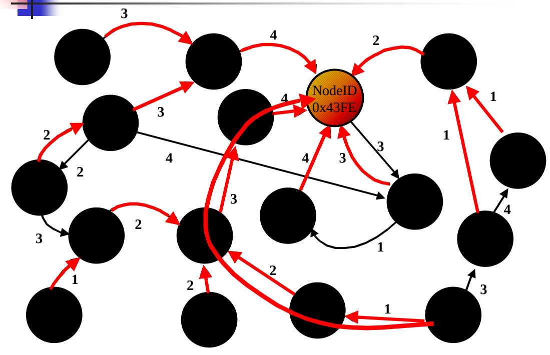
Tapestry

- System developed at Berkeley
  - Motivated by the OceanStore project (a world-wide file system)
- Basics:
  - Similar to CAN and Chord in hashing keys and nodes
  - Key-id space is large (say 2^{160})
  - Interpret Ids has a sequence of digits
  - For example:
    - Key “3AB8” is a key using hex digits
    - Number of digits and size of each digit is customizable

Single Node’s Neighbors

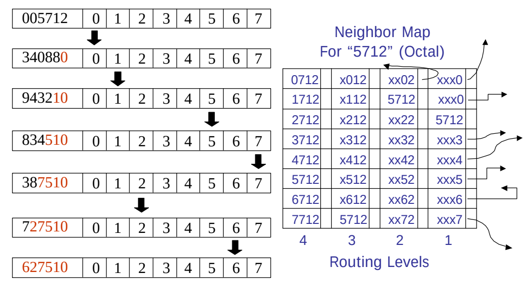
- Neighbors at level “j”:
  - Match suffix for j-1 digits
  - Try to find all possible variations for the jth digit
- For instance, consider node: 0321
  - Level 1 neighbors: 2300, 0321, 1002, 3213
  - Level 2 neighbors: 1201, 1311, 0321, 3231
  - Level 3 neighbors: 2021, 1121, 1221, 0321
  - Level 4 neighbors: 0321, 1321, 2321, 3321

Incremental Suffix-Based Routing

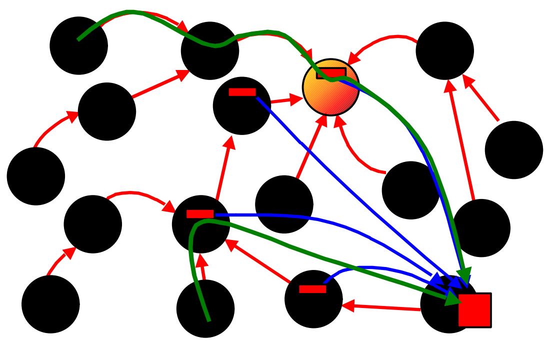


Routing to Nodes

Example: Octal digits, 2^{18} namespace, 005712 → 627510



Object Location: Randomization/Locality



Node addition

- Four components to node addition:
  - "need-to-know" nodes are informed of the new node
  - New node might become the root for existing objects
  - Construct a routing table for this new node
  - Other nodes are informed of this new node and can use this node for routing optimizations
- Step 2 is easy
- Steps 1, 3, 4 require:
  - Limited multicast
  - Probing of nearby nodes