Tapestry & Skip Graphs

Arvind Krishnamurthy
Fall 2003

Tapestry

- Distributed data structure:
  - Maps keys to object locations (pointers)
  - Pointers are wide-area pointers (IP address, port number, etc.)
- Keys interpreted as a sequence of digits
  - Randomly generated
- Incremental suffix routing
  - Source to target route is accomplished by correcting one digit at a time
  - For instance: (to route from 0312 → 1643)
    - 0312 → 2173 → 3243 → 2643 → 1643
  - Each node has a routing table

Routing Table

- Size of routing table:
  Number-levels * digit-range
- Some elements could be missing because of sparse id space
- Assume "n" nodes, octal digits:
  - For level 1, n/8 candidates for each entry
  - For level k, $n/8^k$ candidates for each entry
  - When n=512, there is on average only one candidate for a level 3 neighbor
  - Holes filled with next higher/lower number
  - Degenerate case: self loops

Neighbor Map
For "5712" (Octal)

0712	x012	xx02	xxx0
1712	x112	5712	xxx2
2712	x212	xx22	5712
3712	x312	xx32	xxx3
4712	x412	xx42	xxx4
5712	x512	xx52	xxx5
6712	x612	xx62	xxx6
7712	5712	xx72	xxx7
4	3	2	1

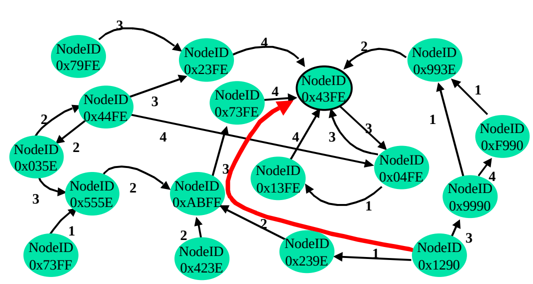
Routing Levels

Publishing

- Node X wants to publish an object O
  - Generates a key for O
  - Routes towards the key in the tapestry system starting with X
  - Deposits a pointer to "X:O" at every routing hop
    - These pointers can be discarded
  - Deposits a permanent pointer at the "root"
    - This pointer cannot be discarded
- Multiple copies of the object could be published at the same time
  - Each deposits pointers along its own trail
  - Trails intersect at the "root"
- Object can be updated without updating the pointers
- Unpublishing objects is hard

Publishing/Locating Objects

Node "1290" wants to publish "53FE"

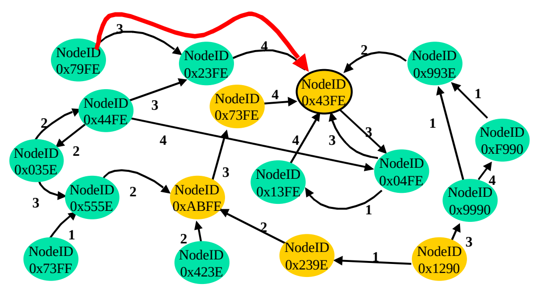


Surrogate Routing

- In the example:
  - "43FE" is considered as the surrogate of "53FE"
- Surrogate is unique irrespective of which node you start searching from
  - As long as there is a consistent rule for finding the surrogate
  - Such as taking the next lower/higher entry in the routing table if the corresponding entry is missing
- System needs to guarantee that neighbor routing tables are kept consistent
  - If "73FE" is missing an entry at level 4 for digit "5"
  - Then "23FE" should also be missing an entry at level 4 for digit "5"

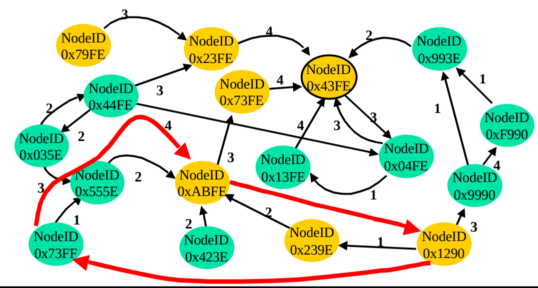
Publishing/Locating Objects

Node "79FE" wants to publish "53FE"



Locating Objects

Node "73FF" wants to find "53FE"



Tapestry Routing Table

- Routing table is optimized to reflect locality
- For instance, consider level 2 for "0234":
  - Let's say we have two candidates for an entry:
    1204, 2704
  - Pick the closest one based on round-trip latency
- Also can have backup entries:
  - Improves fault-tolerance

Node addition

- When a new node joins the network what steps do have to take?

DHT Wrap-up: Common Properties

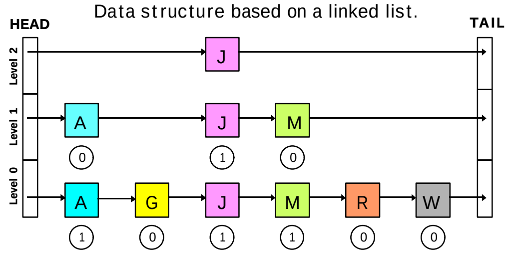
- Underlying metric space.
- Nodes embedded in metric space.
- Location determined by key.
- Hashing to balance load.
- Greedy routing.
- O(log n) space at each node.
- O(log n) routing time.

Announcements

- Begin discussing routing algorithms for Internet/wireless networks on Wednesday
- Friday: guest lecture by Richard Yang
  - Performance of "Selfish routing"

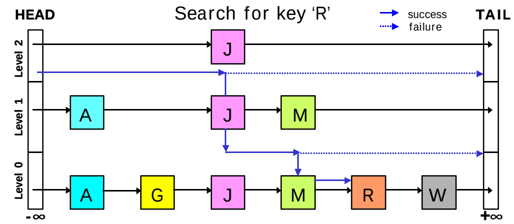
Skip Lists (Pugh '90)

Data structure based on a linked list.



Each node linked at higher level with probability 1/2.

Searching in a Skip List



Time for search: O(log n) on average.
On average, constant number of pointers per node.

Skip lists for P2P?

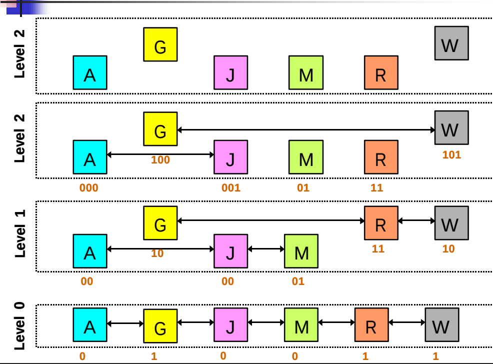
Advantages

- O(log n) expected search time.
- Retains locality.
- Dynamic node additions/deletions.

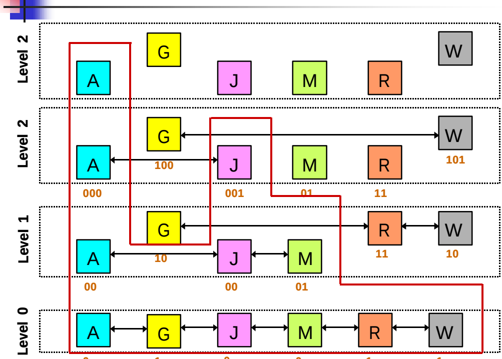
Disadvantages

- Heavily loaded top-level nodes.
- Easily susceptible to random failures.
- Lacks redundancy.

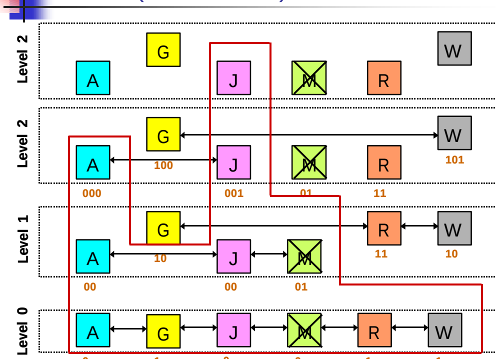
A Skip Graph



Search



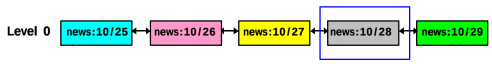
Search (with failures)



Locality and range queries

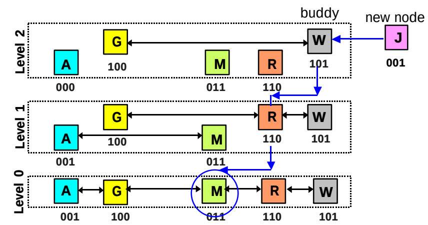
- Find key < F, > F.
- Find largest key < x.
- Find least key > x.
- Find all keys in interval [D..O].

Example: find latest news from yesterday.
→ find largest key < news:10/29.



DHTs cannot do this easily since hashing destroys locality.

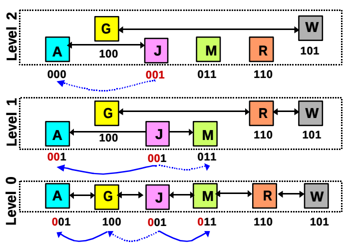
Node Insertion



Starting at buddy node, find nearest key at level 0.
Basically a range query looking for key closest to new key.
Takes O(log n) on average.

Node Insertion (contd.)

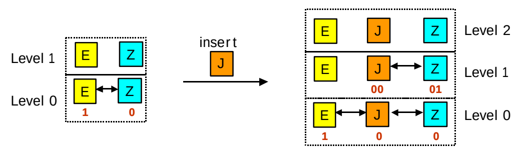
At each level i, find nearest node with matching prefix of membership vector of length i+1.



Total time for insertion: O(log n)
DHTs take: O($\log^2$n)

Independent of system size

No need to know size of keyspace or number of nodes.



Old nodes extend membership vector as required with arrivals.
DHTs require knowledge of keyspace size initially.

Skip Graphs: Discussion

- Virtual to physical node mapping is definitely an issue
- Could use random mapping to guarantee load balance
  - Note that this does not destroy the ability to perform range queries
  - Destroys geographical locality
- State is larger than DHTs:
  - DHT state: O(log n) per node where n is number of nodes
  - Skip graph state: O(log n) per virtual node (with n being number of objects)