Review: DHT Basics

- Objects need unique key
- Key is hashed to integer value
  - Huge key space, e.g. $2^{128}$
- Key space partitioned
  - Each peer gets its key range

DHT Goals
- Efficient routing to the responsible peer
- Efficient routing table maintenance
DHT Design Space

- Minimal routing table
  - Peer state $O(1)$, Avg. path length $O(n)$
  - Brittle network

- Maximal routing table
  - Peer state $O(n)$, Path length $O(1)$
  - Very inefficient routing table maintenance

DHT Routing Tables

- Usual routing table
  - Peer state $O(\log n)$, Path length $O(\log n)$
  - Compromise between routing efficiency and maintenance efficiency
DHT Algorithms

- CHORD
- Pastry
- Symphony
- Viceroy
- CAN

Pastry: Basics

- 128 bit circular id space
- Routing table elements
  - Leaf set: Key space proximity
  - Routing table: long distance links
  - Neighborhood set: network proximity
- Basic routing
  
  If (target key in key space proximity)
  
    Use direct leaf set link
  
  else
  
    Use link from routing table to resolve next digit of target key
Pastry: Leaf sets

- Each node maintains IP addresses of the nodes with the L numerically closest larger and smaller node IDs, respectively.
  - routing efficiency/robustness
  - fault detection (keep-alive)
  - application-specific local coordination

Pastry: Routing table

- L nodes in leaf set
- $\log_2 b$ N Rows
  - (actually $\log_2 b 2^{128} = 128/b$)
- $2^b$ columns
- L network neighbors

<table>
<thead>
<tr>
<th>Nodeld 10233102</th>
<th>Leaf set</th>
<th>SMALLER</th>
<th>LARGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10233003</td>
<td>10233021</td>
<td>10233120</td>
<td>10233122</td>
</tr>
<tr>
<td>10233001</td>
<td>10233000</td>
<td>10233200</td>
<td>10233222</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Routing table</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0-221202 1</td>
</tr>
<tr>
<td>1-1-301203 1-2-220203 1-3-120203</td>
</tr>
<tr>
<td>10-0-1203 10-1-32102 10-2-32302</td>
</tr>
<tr>
<td>10-2-0220 10-2-1-302 10-2-2-2302 10-2-3-2302</td>
</tr>
<tr>
<td>1023-0-32 1023-1-000 1023-2-121 3</td>
</tr>
<tr>
<td>1023-3-0-61 1023-3-2-32</td>
</tr>
<tr>
<td>0 102333-2-0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Neighborhood set</th>
</tr>
</thead>
<tbody>
<tr>
<td>13021022 10200230 11301233 31301233</td>
</tr>
<tr>
<td>02212102 22301203 31203203 35213321</td>
</tr>
</tbody>
</table>
Pastry: Routing

- \( \log_2 N \) steps
- \( O(\log N) \) state

### Pastry: Routing procedure

If (destination is within range of our leaf set)
- forward to numerically closest member

else
  - let \( l \) = length of shared prefix
  - let \( d \) = value of \( l \)-th digit in \( D \)'s address
  - if \( (R_i^d \text{ exists}) \)
    - forward to \( R_i^d \)
  - else
    - forward to a known node* that
      - (a) shares at least as long a prefix
      - (b) is numerically closer than this node

*from LeafSet, RoutingTable, or NetworkNeighbors
**Pastry: Routing Properties**

- **O(log N) routing table size**
  - $2b \cdot \log b N + 2l$

- **O(log N) message forwarding steps**

- **Network stability:**
  - Guaranteed unless $L/2$ simultaneous failures of nodes with adjacent nodeIds

- **Number of routing hops:**
  - No failures: $< \log_2 b N$ average, $128/b + 1$ max
  - During failure recovery $O(N)$ worst case, average case much better

---

**Pastry: Node addition**

- New node: $X = d46a1c$
- $A = 65a1fc$
- Route($d46a1c$)
- $Z = d467c4$
- $d462ba$
- $d4213f$
- $d13da3$
Routing table maintenance

• Leaf set
  ▶ Copy from neighbor
  ▶ Extend by sending request to right/left boundary leaf link

• Routing table
  ▶ Collect routing tables from peers encountered during network entry
    ■ Works because peers encountered share same prefix
  ▶ Can be incomplete

• Network neighbor set
  ▶ Probe nodes from collected routing tables
  ▶ Request neighborhood sets for known nearby nodes

Pastry: Locality properties

• Assumption: scalar proximity metric
  ▶ e.g. ping/RTT delay, # IP hops, geographical distance
  ▶ a node can probe distance to any other node

• Proximity invariant:
  ▶ Each routing table entry refers to a node close to the local node (in the proximity space), among all nodes with the appropriate nodeid prefix.
Pastry: Geometric Routing in proximity space

- Network distance for each routing step is exponentially increasing (entry in row $i$ is chosen from a set of nodes of size $N/2^i$)
- Distance increases monotonically (message takes larger and larger strides)

Pastry: Locality properties

- Each routing step is local, but there is no guarantee of globally shortest path
- Nevertheless, simulations show:
  - Expected distance traveled by a message in the proximity space is within a small constant of the minimum
- Among $k$ nodes with nodeIds closest to the key, message likely to reach the node closest to the source node first
Pastry: Node addition details

- New node X contacts nearby node A
- A routes “join” message to X, which arrives to Z, closest to X
- X obtains leaf set from Z, i’th row for routing table from i’th node from A to Z
- X informs any nodes that need to be aware of its arrival

Node departure/failure

- Leaf set repair (eager – all the time):
  - Leaf set members exchange keep-alive messages
  - request set from furthest live node in set
- Routing table repair (lazy – upon failure):
  - get table from peers in the same row, if not found – from higher rows
- Neighborhood set repair (eager)
Pastry: Average # of hops

- Average number of hops:
  - Pastry
  - Log(N)

Pastry distance vs IP distance

- Mean = 1.59
**Pastry Summary**

- **Usual DHT scalability**
  - Peer state $\log(N)$
  - Avg. path length $\log(N)$

- **Very robust**
  - Different routes possible
  - Lazy routing table update sufficient

- **Network proximity aware**
  - No IP network detours

**Symphony**

- **Symphony DHT**
  - Map the nodes and keys to the ring
  - Link every node with its successor and predecessor
  - Add $k$ random links with probability proportional to $1/(d \cdot \log N)$, where $d$ is the distance on the ring
  - Lookup time $O(\log^2 N)$
  - If $k = \log N$ lookup time $O(\log N)$
  - Easy to insert and remove nodes (perform periodical refreshes for the links)
**Symphony in a Nutshell**

- Nodes arranged in a *unit circle* (perimeter = 1)
- Arrival --> Node chooses position along circle uniformly at random
- Each node has 1 short link (next node on circle) and *k* long links

Adaptation of Small World Idea: [Kleinberg00]

Long links chosen from a probability distribution function: $p(x) = \frac{1}{x \log n}$ where $n = \#\text{nodes}$

Simple greedy routing:

"Forward along that link that minimizes the absolute distance to the destination."

Average lookup latency = $O(\log^2 n) / k$ hops

Fault Tolerance:
- No backups for long links! Only short links are fortified for fault tolerance.

---

**Network Size Estimation Protocol**

Problem: What is the current value of $n$, the total number of nodes?

$x = \text{Length of arc}$

$\frac{1}{x} = \text{Estimate of } n$

3 arcs are enough.
**Step 0: Symphony**

- Probability Distribution:
  \[ p(x) = \frac{1}{x \log n} \]

  **Symphony:**
  “Draw from the PDF k times”

**Step 1: Step-Symphony**

- Probability Distribution:
  \[ p(x) = \frac{1}{x \log n} \]

  **Step-Symphony:**
  “Draw from the discretized PDF k times”
Step 2: Divide PDF into $\log n$ Equal Bins

**Step-Partitioned-Symphony:**
“Draw exactly once from each of $k$ bins”

Step 3: Discrete PDF

**Chord:**
“Draw exactly once from each of $\log n$ bins”
Each bin is essentially a point.
Two Optimizations

- **Bi-directional Routing**
  - Exploit both outgoing and incoming links!
  - Route to the neighbor that minimizes absolute distance to destination
  - Reduces avg. latency by 25-30%

- **1-Lookahead**
  - List of neighbor’s neighbors
  - Reduces avg. latency by 40%

- Also applicable to other DHTs

---

Symphony: Summary (1)

- **Distributed Hashing in a Small World**
- **Like Chord:**
  - Overlay structure: ring
  - Key ID space partitioning
- **Unlike Chord:**
  - Routing Table
    - Two short links for immediate neighbors
    - $k$ long distance links for jumping
    - Long distance links are built in a probabilistic way
    - Peers are selected using a Probability Distribution Function (pdf)
    - Exploit the characteristics of a small-world network
  - Dynamically estimate the current system size
### Symphony: Summary (2)

- Each node has $k = O(1)$ long distance links
  - Lookup:
    - Expected path length: $O((\log^2 N)/k)$ hops
  - Join & leave
    - Expected: $O(\log^2 N)$ messages
- **Comparing with Chord:**
  - Discard the strong requirements on the routing table (finger table)
  - Rely on the small world to reach the destination.

---

### Viceroy network

- Arrange nodes and keys on a ring
  - As usual
Viceroy network

- Assign to each node a level value
  - chosen uniformly from the set \{1, \ldots, \log n\}
  - estimate \( n \) by taking the inverse of the distance of the node with its successor
  - easy to update

Viceroy network

- Create a ring of nodes within the same level
**Downward links**

- For peer with key x at level i
  - Direct successor peer on level $i+1$
  - Long link to peer $x+2i$ on level $i+1$

**Upward links**

- For each peer with key x at level i
  - Predecessor link on level $i-1$
  - Long link to peer at $x-2i$ on level $i-1$
**Butterfly links**

- Each node $x$ at level $i$ has two **downward** links to level $i+1$
  - a left link to the first node of level $i+1$ after position $x$ on the ring
  - a right link to the first node of level $i+1$ after position $x + (\frac{1}{2})^i$

**Viceroy**

- **Emulating the butterfly network**

- Logarithmic path lengths between any two nodes in the network
- Constant degree per node
**Viceroy Summary**

- **Scalability: Optimal peer state**
  - Peer state $\log(1)$
  - Avg. path length $\log(N)$

- **Complex algorithm**
- **Network proximity not taken into account**

---

**CAN: Overview**

- **Early and successful algorithm**
- **Simple & elegant**
  - Intuitively to understand and implement
  - many improvements and optimizations exist
  - Sylvia Ratnasamy et al. in 2001

- **Main responsibilities:**
  - CAN is a distributed system that maps keys onto values
  - Keys hashed into $d$ dimensional space
  - Interface:
    - `insert(key, value)`
    - `retrive(key)`
CAN

- Virtual $d$-dimensional Cartesian coordinate system on a $d$-torus
  - Example: 2-d $[0,1] \times [0,1]$  
- Dynamically partitioned among all nodes
- Pair $(K,V)$ is stored by mapping key $K$ to a point $P$ in the space using a uniform hash function and storing $(K,V)$ at the node in the zone containing $P$
- Retrieve entry $(K,V)$ by applying the same hash function to map $K$ to $P$ and retrieve entry from node in zone containing $P$
  - If $P$ is not contained in the zone of the requesting node or its neighboring zones, route request to neighbor node in zone nearest $P$

State of the system at time $t$

In this 2-dimensional space a key is mapped to a point $(x,y)$
**CAN: Routing**

- d-dimensional space with n zones
- 2 zones are neighbours if d-1 dimensions overlap

Algorithm:
Choose the neighbor nearest to the destination

**CAN: Construction - Basic Idea**
1) Discover some node “I” already in CAN
2) Pick random point in space

3) I routes to (x,y), discovers node J
**CAN: Construction**

4) split J’s zone in half... new owns one half

---

**CAN-Improvement: Multiple Realities**

- Build several CAN-networks
- Each network is called a reality
- Routing
  - Jump between realities
  - Chose reality in which distance is shortest
**CAN-Improvement: Multiple Dimensions**

![Graph showing number of hops vs. number of nodes for different dimensions](image)

- **More dimensions**: shorter paths
- **More realities**: more robustness

**CAN: Multiple Dimensions vs. Multiple Realities**

- **More dimensions**: shorter paths
- **More realities**: more robustness
- **Trade-off?**
**CAN: Summary**

- Inferior scalability
  - Peer state $O(d)$
  - Avg. path length $O(d N^{1/d})$

- Useful for spatial data!

---

**Spectrum of DHT Protocols**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>#links</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deterministic Topology</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>$O(d)$</td>
<td>$O(d N^{1/d})$</td>
</tr>
<tr>
<td>Chord</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td><strong>Partly Randomized Topology</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Viceroy</td>
<td>$O(1)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Pastry</td>
<td>$O((2^b-1)/(\log N/b))$</td>
<td>$O((\log N) / b)$</td>
</tr>
<tr>
<td><strong>Completely Randomized Topology</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symphony</td>
<td>$O(k)$</td>
<td>$O((\log^2 N)/k)$</td>
</tr>
</tbody>
</table>
Latency vs State Maintenance

Network size: \( n = 2^{15} \) nodes

Distributed Hash Table Algorithms

L3S Research Center