CS 244b - Coral

Partitioning & scalability
Consistent Hashing
P2P Systems
Non-linearizable Storage Systems
$H(\text{dom}) \rightarrow a \ 160\text{-bit SHA-1}$
\[ H(\text{server ID}, \text{URL}) \to h_i \]

\[ h_1 \text{ /capacity} \]

\[ h_2 \]

CARP
\[ b_0, b_1, b_2, \ldots \]

\[ 2x, 2x+1 \]

KOORDE
2004

No EC2

P2P, DHT

Straw man 1: $k = H(\text{URL})$, $v = \text{web page}$
use Kademlia to store $(k, v)$

Straw man 2: network of web proxies
use KAD for $k = H(\text{URL})$, $v = \text{proxy addr.}$
- Query far away nodes to find nearby data

  Join multiple DHTs - diameter

  Stop put at full loaded node

- Download from nearby nodes
- Map clients to nearby Coral Proxies
  - Probe DNS resolver

  DNSName *nyuud.net ➔
  *172.70.nyuud.net

  OASIS
- \textit{put(key, val, ttl, [levels])}: Inserts a mapping from the key to some arbitrary value, specifying the time-to-live of the reference. The caller may optionally specify a subset of the cluster hierarchy to restrict the operation to certain levels.

- \textit{get(key, [levels])}: Retrieves some subset of the values stored under a key. Again, one can optionally specify a subset of the cluster hierarchy.

- \textit{nodes(level, count, [target], [services])}: Returns \textit{count} neighbors belonging to the node’s cluster as specified by \textit{level}. \textit{target}, if supplied, specifies the IP address of a machine to which the returned nodes would ideally be near. Coral can probe \textit{target} and exploit network topology hints stored in the DSHT to satisfy the request. If \textit{services} is specified, Coral will only return nodes running the particular service, \textit{e.g.}, an HTTP proxy or DNS server.

- \textit{levels()}: Returns the number of levels in Coral’s hierarchy and their corresponding RTT thresholds.
- Avoid hotspots in canal
  Don't take shortcuts in routing
Figure 4: The number of client accesses to CoralProxies and the origin HTTP server. CoralProxy accesses are reported relative to the cluster level from which data was fetched, and do not include requests handled through local caches.
<table>
<thead>
<tr>
<th>Request latency (sec)</th>
<th>All nodes</th>
<th></th>
<th>Asian nodes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50%</td>
<td>96%</td>
<td>50%</td>
<td>96%</td>
</tr>
<tr>
<td>single-level</td>
<td>0.79</td>
<td>9.54</td>
<td>2.52</td>
<td>8.01</td>
</tr>
<tr>
<td>multi-level</td>
<td>0.31</td>
<td>4.17</td>
<td>0.04</td>
<td>4.16</td>
</tr>
<tr>
<td>multi-level, traceroute</td>
<td>0.19</td>
<td>2.50</td>
<td>0.03</td>
<td>1.75</td>
</tr>
</tbody>
</table>

**Figure 5:** End-to-End client latency for requests for Coralized URLs, comparing the effect of single-level vs. multi-level clusters and of using traceroute during DNS redirection. The top graph includes all nodes; the bottom only nodes in Asia.
Figure 6: Latencies for proxy to \textit{get} keys from Coral.
Figure 8: The total number of *put* RPCs hitting each Coral node per minute, sorted by distance from node ID to target key.