Techniques for Efficient Routing and Load Balancing in Content-Addressable Networks

Ozgur D. Sahin
Divyakant Agrawal
Amr El Abbadi

Department of Computer Science
University of California at Santa Barbara
Motivation

- Distributed Hash Tables (DHTs) effectively support exact key lookups.
- Emerging applications that harness the power of DHTs to support more functionality:
  - Content-based lookup, event dissemination, range queries, etc.
- Content-Addressable Network (CAN) is appealing for such applications as its multi-dimensional logical space enables direct mapping of multi-dimensional data.
Semantic-CAN Applications

- **pSearch**: Document similarity search
  - CAN dimensions are the ‘concepts’

- **Meghdoot**: Publish/subscribe system over P2P
  - CAN dimensions are event attributes
Challenges

Mapping data items based on their values introduces load imbalance
  - No hashing is used
  - Real data sets are usually skewed

Thus semantic-CAN applications should deal with two problems:
  - Load imbalance due to value-based mapping
  - Inefficient routing in CAN (especially in low dimensions) compared to other DHTs
Outline

- Distributed Hash Tables (DHTs)
  - Content-Addressable Networks (CAN)
- Improvements for CAN
  - PART I - Improving Routing
    - Long-Distance Pointers: *Random and Subspace*
    - Evaluation
  - PART II - Load Balancing
    - *Multiple Join, Forwarded Join, Multiple Zones*
    - Evaluation
- Conclusions and Future Work
Distributed Hash Tables (DHTs)

- Implement a hashtable-like interface in a distributed system
- Support efficient routing and exact lookups
- Objects are hashed onto a logical space, which is dynamically partitioned among peers

Example DHTs:
- CAN: multi-dimensional Cartesian space
- Chord: Identifier ring
- Tapestry, Pastry: Identifier mesh
Uses a $d$-dimensional Cartesian space

Space is partitioned into rectangular zones

Each zone is maintained by a peer in the system

Each object is hashed to a point and assigned to the corresponding peer

2-dimensional CAN

object $O$
Each peer keeps routing information about its neighbors in logical space:
- IP address and zone coordinates

Routing is greedy: Each peer passes the message to the neighbor closest to destination.

B’s neighbors = \{A, C, D\}
When a new peer joins:
  - Sends a ‘join’ message to a random point
  - Corresponding peer assigns half of its zone to new peer
  - Affected neighbor lists are updated
PART I – Improving Routing

- CAN routing is $O(N^{1/d})$ with $O(d)$ routing state per node
  - Advantage: Constant neighbor state
  - Disadvantage: Routing is inefficient especially when $d$ is small (compared to other DHTs that provide logarithmic routing using logarithmic routing state per node)
Long Distance Pointers (LDP)

- **Observation:** CAN only uses immediate neighbors during routing. There is no way of making ‘big jumps’ in the logical space.

- **Solution:** Keep a few long distance pointers (LDPs) in addition to regular CAN neighbors.
  - Peers consider both CAN neighbors and LDPs during routing.
  - More efficient routing at the expense of keeping more routing state at the peers.
LDP Schemes

- **Random Pointers:** Select LDPs randomly
- **Subspace Pointers:** Select one LDP from each sub-region
  - Better space coverage than *random scheme*
Advantages of LDPs

- Correct message delivery is guaranteed by the regular CAN neighbors. LDPs are only for efficiency.
- Number of LDPs can be set locally based on resources and requirements
- LDPs can be maintained lazily
  - LDPs are only refreshed when they are found to be inaccessible
  - Not affected directly by peer join and leaves
Evaluation of LDP Schemes

- CAN simulator in Java
- **By default:** 16K peers, CAN dimensionality is 2, number of LDPs is 16
- Measure average routing cost
  - Number of hops visited to route a message between two arbitrary points
  - Results are averaged over $10^5$ runs
- Varied: 1) dimensionality of CAN, 2) Number of LDPs, 3) Number of peers
Both LDP schemes greatly improve the routing performance of CAN (especially in low dimensions)

Subspace scheme is slightly better than Random scheme
Routing Cost vs. LDP#

- Even a single LDP reduces the cost more than 50%
- 8 to 32 LDPs provides reasonable improvement
- With 64 LDPs, CAN outperforms Chord

Routing cost is **50.22** for CAN

<table>
<thead>
<tr>
<th>LDP #</th>
<th>Random</th>
<th>Subspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22.12</td>
<td>22.12</td>
</tr>
<tr>
<td>2</td>
<td>18.02</td>
<td>17.77</td>
</tr>
<tr>
<td>4</td>
<td>14.58</td>
<td>14.45</td>
</tr>
<tr>
<td>8</td>
<td>11.83</td>
<td>11.64</td>
</tr>
<tr>
<td>16</td>
<td>9.60</td>
<td>9.41</td>
</tr>
<tr>
<td>32</td>
<td>7.81</td>
<td>7.64</td>
</tr>
<tr>
<td>64</td>
<td>6.35</td>
<td>6.16</td>
</tr>
<tr>
<td>128</td>
<td>5.10</td>
<td>4.93</td>
</tr>
</tbody>
</table>
Routing Cost vs. Peer#

Routing cost is smaller and increases slowly with LDPs:
When peer number changes from 4K to 64K:
  4x increase for CAN, 2.5x increase for Subspace scheme
Part II – Load Balancing

- Mapping of data items based on their values introduces load imbalance in the system.
- Existing load balancing algorithms for DHTs are not directly applicable to semantic-CAN applications.

**Solution:**
- Load-aware peer joins
- Dynamic monitoring of load distribution and ‘load redistribution’ when necessary.
Load Balancing Schemes

- **Multiple Join**: Contact multiple peers during join and select the one with the highest load.

- **Forwarded Join**: Peers keep load information about some other peers and forward incoming join requests to more loaded peers.
  - Keep neighbor load or maintain a separate index (Skip Graph) on load values.

Above two schemes can only be executed at join time!
Load Balancing Schemes

Multiple Zones: Load-aware joins as in ‘Forwarded Join’); but also each peer periodically compares its load with the highest load it knows of. If the ratio is smaller than a threshold $T_L$, then it hands over its zone to a neighbor and splits the loaded peer.

A peer might maintain multiple zones.

Peers check to see if their zones can be merged with neighbors to avoid fragmentation.
Evaluation of LB Schemes

- CAN simulator with 16K peers
- 12 data items inserted for each peer (data values are normally distributed)

Configuration:

- **Multiple Join**: a new peer contacts 4 peers
- **Forwarded Join**: assume a new peer splits the zone of the highest loaded peer
- **Multiple Zones**: $T_L = 0.4$ and each peer executes the dynamic load check once after all peers join
Load Distribution in 2D

(There were 1140 additional zones for Multiple Zones scheme)
Routing Cost with Load Balancing

- LB schemes increase the routing cost
- Routing performance for *Multiple Zones* scheme can be improved to **50.32** with routing improvement

<table>
<thead>
<tr>
<th></th>
<th>Routing Cost</th>
<th>Avg. Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>50.22</td>
<td>4.58</td>
</tr>
<tr>
<td>Multiple Join</td>
<td>56.34</td>
<td>4.38</td>
</tr>
<tr>
<td>Forwarded Join</td>
<td>59.69</td>
<td>4.30</td>
</tr>
<tr>
<td>Multiple Zones</td>
<td>61.39</td>
<td>4.29</td>
</tr>
</tbody>
</table>
Load Distribution in 6D

16K Peers, 6 Dimensions

Percentage of Indices

Percentage of Peers

- CAN
- Multiple Join
- Forwarded Join
- Multiple Zones
Conclusions

- We showed that it is possible to achieve both efficient routing and load balancing in CAN.
- Routing performance can be improved significantly by keeping a few LDPs.
- Selecting LDPs from different sub-regions provides better results than selecting them randomly.
- Load balancing can be achieved by load aware joins and dynamic load redistribution.
- Load balancing results in a slightly reduced routing performance.
Future Work

- Design additional LDP schemes that are adaptive to the routing patterns of the peers
- Integrate the processing load and non-regular partitioning into the load balancing schemes
- Evaluate the performance when both routing and load balancing schemes are used
Thank you…

Questions?

GAIA Project
www.cs.ucsb.edu/~dsl/gaia