A Scalable Content Addressable Network

Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001

Presented by: Maryam Moharramzadeh
Supervised by: Dr. Ahmet Burak CAN
Outline

1. Introduction
2. Basic Design
3. Advanced Architecture
4. Applications
5. Discussion

A Scalable Content-Addressable Network
Simplified algorithm key features (reminder)

- The Key-Value entries of the hash table are distributed between the CAN nodes (each node stores a fraction of the table).
- The coordinate space is completely logical, not related to the underlying physical (IP) network.
- An entry \((K_1, V_1)\) is mapped to a single point \(P\) in the coordinate space, and stored at the node which owns the zone in which \(P\) lies.
- To retrieve an entry corresponding to key \(K_1\), the same hash function is applied to map \(K_1\) to \(P\), and a query is routed to the node in whose zone \(P\) lies.
- Each CAN node stores a routing table containing IP addresses and virtual coordinate zone of each of its neighbors (which zones are adjacent to its own).
CAN evaluation metrics

- **Path length**
  - The average number of (application-level) hops required to route between two points in the coordinate space.

- **Neighbor state**
  - The number of neighbors every CAN node must be “familiar” with.

- **Latency**
  - Average end-to-end latency of the total routing path between two points in the coordinate space.
  - Average latency of a single hop between adjacent nodes in the CAN overlay network.

- **Zones Volume**
  - An indication of the requests and storage load a node must handle.

- **Routing fault tolerance**
  - The availability of multiple paths between two points in the CAN.

- **Hash table availability**
  - Adequate replication of a (key, value) entry to withstand the loss of one or more nodes.
Design Improvements

- Tradeoff considerations

- Improved routing performance and system robustness. \textbf{VS.} Increased complexity and per-node state.

- Requires further studying and deployment experience.
Design Improvements

Proposed improvements

1. Multi-dimensional coordinate space
2. Realities: multiple coordinate spaces
3. Overloading coordinate zone
4. Multiple hash functions
5. More Uniform Partitioning
6. Caching and Replication techniques for “hot spot” management
Multi-dimensional coordinate space

- Observation 1: The design doesn’t restrict the dimensionality of the coordinate space.
- Increasing the dimensionality of the coordinate space reduces the routing path length, and hence the path latency.
- Since increasing the number of dimensions implies that a node has more neighbors, the routing fault tolerance also improves.
  - As a node now has more potential next hop nodes, along which messages can be routed in the event that one or more neighboring nodes crash.
  - The price- increased per-node state, and maintenance traffic.
Multi-dimensional coordinate space

2D space

3D space

Reminder: The dimensions are logical, not physical.
Multi-dimensional coordinate space

The effect of increasing dimensionality on the length of an average path in CAN
## Design Improvements

- **Proposed improvements**
  1. Multi-dimensional coordinate space
  2. **Realities: multiple coordinate spaces**
  3. Overloading coordinate zone
  4. Multiple hash functions
  5. More Uniform Partitioning
  6. Caching and Replication techniques for “hot spot” management
Observation 2: It is possible to maintain multiple, independent coordinate spaces, with each node in the system being assigned a different zone in each coordinate space.

Each such coordinate space is called a “reality”.

The contents of the hash table are replicated on every reality.

Hence, for a CAN with \( r \) realities, a single node is assigned \( r \) coordinate zones, and hold \( r \) independent neighbors set.

what’s the size of neighbor state in a system with \( d \) dimensions and \( r \) realities?
Realities: multiple coordinate spaces

**Reality 1**

- A
- D
- C
- F
- E
- G
- B
- H

Zone_{x,y}
Realities: multiple coordinate spaces

Reality 2

Zone_{a,b}
Realities: multiple coordinate spaces

Realities 1&2

A Scalable Content-Addressable Network
Realities: multiple coordinate spaces - Pros

- **Improved data availability**
  - For example, say an entry is to be stored at the coordinate location \((x, y, z)\). With 4 independent realities, this entry would be stored at 4 different nodes corresponding to the coordinates \((x, y, z)\) on each reality and hence it is unavailable only when all 4 nodes are unavailable.

- **Improved routing fault tolerance**
  - In the case of a routing breakdown in one reality, messages can continue to be routed using the remaining realities.

- **Reduced path length**
  - Observation: Routing to location \((x, y, z)\) translates to reaching \((x, y, z)\) on any reality.
  - A given node owns one zone per reality, each of which is at a distinct, and possibly distant, location in the coordinate space.
  - To forward a message, a node now checks all its neighbors on each reality and forwards the message to that neighbor with coordinates closest to the destination.
Realities: multiple coordinate spaces

The effect of increasing realities on the length of an average path in CAN

![Graph showing the effect of increasing realities on the length of an average path in CAN.](image)
Multiple dimensions VS. multiple realities

- Both improvements resemble:
  - Both result in **shorter path lengths**.
  - Both increase the per-node neighbor state and maintenance traffic.

- An evaluation shows that the same number of neighbors, increasing the dimensions of the space yields **shorter path lengths** than increasing the number of realities.

- Multiple realities offer other benefits such as improved **data availability** and **fault-tolerance**.
Multiple dimensions VS. multiple realities

A comparison of the effect of dimensions/realities on path length

![Graph showing the effect of dimensions and realties on path length. The graph illustrates the number of hops versus the number of neighbors for different values of dimensions (d) and realties (r). The number of nodes is 131,072.]
Design Improvements

- Proposed improvements
  1. Multi-dimensional coordinate space
  2. Realities: multiple coordinate spaces
  3. Overloading coordinate zone
  4. Multiple hash functions
  5. More Uniform Partitioning
  6. Caching and Replication techniques for “hot spot” management
Overloading coordinate zones

- So far, the design assumes that a zone is, at any point in time, assigned to a single node in the system.
- We now modify this to allow multiple nodes to share the same zone. Nodes that share the same zone are termed peers.
- A node maintains a list of its peers in addition to its neighbor list.
- While a node must know all the peers in its own zone, it need not track all the peers in its neighboring zones. Rather, a node elects one neighbor from amongst the peers in each of its neighboring zones.
Overloading coordinate zones

Every peer chooses the closest peer from the neighboring zone.
Overloading coordinate zones - the algorithm

- When a new node $A$ joins the system, it discovers an existing (already in the CAN) node $B$ whose zone it is meant to occupy.

- Rather than directly splitting its zone as described earlier, node $B$ first checks whether it has fewer than $\text{MAXPEERS}$ peer nodes:
  
  - Yes - $A$ merely joins $B$'s zone without any space splitting. Node $A$ obtains both its peer list and its list of coordinate neighbors from $B$.
  
  - No - then the zone is split into half as before:
    
    - Node $B$ informs each of the nodes on it's peer-list that the space is to be split.
    
    - Using a deterministic rule the nodes on the peer list together with the new node $A$ divide themselves equally between the two halves of the now split zone.
Overloading coordinate zones - the algorithm

- Periodically, a node sends to its coordinate neighbors a request for its list of peers.

- The contents of the hash table itself may be either divided or replicated across the nodes in a zone.
Overloading coordinate zones - Pros & Cons

- **Reduced path length and hence reduced path latency**
  - Since placing multiple nodes per zone has the same effect as reducing the number of nodes in the system.

- **Reduced per-hop latency**
  - Since a node now has multiple choices in its selection of neighboring nodes and can select neighbors that are closer in terms of latency.
  - We see that placing 4 nodes per zone can reduce the per-hop latency by about 45%.

- **Improved fault tolerance**
  - Since a zone is vacant only when all the nodes in a zone crash simultaneously.

- **On the negative side, overloading zones adds somewhat to system complexity because nodes must additionally track its set of peers, in addition to its set of neighbors.**

<table>
<thead>
<tr>
<th>Number of nodes per zone</th>
<th>per-hop latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>116.4</td>
</tr>
<tr>
<td>2</td>
<td>92.8</td>
</tr>
<tr>
<td>3</td>
<td>72.9</td>
</tr>
<tr>
<td>4</td>
<td>64.4</td>
</tr>
</tbody>
</table>
Design Improvements

- **Proposed improvements**
  1. Multi-dimensional coordinate space
  2. Realities: multiple coordinate spaces
  3. Overloading coordinate zone
  4. **Multiple hash functions**
  5. More Uniform Partitioning
  6. Caching and Replication techniques for “hot spot” management
Multiple hash function

- $k$ different hash functions to map a single key onto $k$ points in the coordinate space.
- Accordingly replicate a single (key, value) pair at $k$ distinct nodes in the system.
- Improves data availability.
- To improve latency:
  - Assume node $A$ wants to send a query to the CAN, i.e. retrieve the value associated to some key $K_1$.
  - Since $K_1$ is mapped to $k$ different nodes (by each of the hash functions), the node forwards the query to the closest.
Multiple hash function

An example for $k=3$

The key $K_1$ is mapped to 3 nodes by the hash functions $h_1, h_2, h_3$. 

![Diagram showing hash function mapping]
Design Improvements

Proposed improvements

1. Multi-dimensional coordinate space
2. Realities: multiple coordinate spaces
3. Multiple hash functions
4. More Uniform Partitioning
5. Caching and Replication techniques for “hot spot” management
More Uniform Partitioning

- Since (key, value) pairs are spread across the coordinate space using a uniform hash function, the volume of a node’s zone is indicative of the size of the (key, value) database the node will have to store.

- A uniform partitioning of the space is thus desirable to achieve adequate load balancing.
Design Improvements

- Proposed improvements
  1. Multi-dimensional coordinate space
  2. Realities: multiple coordinate spaces
  3. Overloading coordinate zone
  4. Multiple hash functions
  5. More Uniform Partitioning
  6. Caching and Replication techniques for “hot spot” management
Caching and Replication techniques for “hot spot” management

- As with files in the Web, certain (key, value) pairs in a CAN are likely to be far more frequently accessed than others, thus **overloading nodes that hold these popular data keys**.

- To make very popular data keys widely available, we borrow some of the caching and replication techniques commonly applied to the Web:
  - **Caching**: a CAN node maintains a cache of the data keys it recently accessed.
  - **Replication**: A node that finds it is being overloaded by requests for a particular data key can replicate the data key at each of its neighboring nodes.
    - A popular data key is thus eventually replicated within a region surrounding the original storage node.
    - A node holding a replica of a requested data key can, with a certain probability, choose to either satisfy the request or forward it on its way, thereby causing the load to be spread over the entire region rather than just along the periphery.
Design Improvements - Summary

- Path latency
  - Multiple dimensions
  - Multiple realities
  - RTT weighted metric
  - Overloading coordinate zones
  - Multiple hash functions
  - Topology-sensitive construction

- Routing robustness
  - Multiple dimensions
  - Multiple realities
  - Overloading coordinate zones
  - Multiple hash functions

- Data availability
  - Multiple realities
  - Multiple hash functions

- Load balancing
  - More uniform partitioning.
  - Caching and replication.
Design Improvements - Summary

To Measure the cumulative effect of all the above features, 2 algorithms were compared:

1. The basic simplified algorithm - “bare bone”.
2. “Knobs on full” algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>“bare bones” CAN</th>
<th>“knobs on full” CAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>$r$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$p$</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>$k$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RTT weighted routing metric</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>Uniform partitioning</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>Landmark ordering</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Why is the IP latency lower?

<table>
<thead>
<tr>
<th>Metric</th>
<th>“bare bones” CAN</th>
<th>“Knobs on full CAN”</th>
</tr>
</thead>
<tbody>
<tr>
<td>path length</td>
<td>198.0</td>
<td>5.0</td>
</tr>
<tr>
<td># neighbors</td>
<td>4.57</td>
<td>27.1</td>
</tr>
<tr>
<td># peers</td>
<td>0</td>
<td>2.95</td>
</tr>
<tr>
<td>IP latency</td>
<td>115.9ms</td>
<td>82.4ms</td>
</tr>
<tr>
<td>CAN path latency</td>
<td>23,008ms</td>
<td>135.29ms</td>
</tr>
</tbody>
</table>

Table 4: CAN parameters

Table 5: CAN Performance Results
Outline

1. Introduction
2. Basic Design
3. Advanced Architecture
4. Applications
5. Discussion
Applications

- We were unable to find any actual applications of CAN
  - all the schemes described in the literature we’ve reviewed are solely experimental.

- Why?
  - Companies keep their system architecture in secret?
  - The scientific community is occupied with non-practical research?
Outline

1. Introduction
2. Basic Design
3. Advanced Architecture
4. Applications & Citations
5. Discussion
The work addresses two key problems in the design of Content-Addressable networks
- Scalable routing.
- Scalable indexing.

Evaluation validates the scalability
- A simulated system with over 260,000 randomly generated nodes.
- The latency was less than 2 times the latency of the underlying IP network.

Security aspects should be considered
- DOS resistant CAN.
- A malicious node can act, not only as a malicious client, but also as a malicious server or router.
Q&A

Any Questions?