CONTENT ADDRESSABLE NETWORKS (CAN)

A content-addressable network (CAN) is scalable indexing mechanism that maps objects to their locations in the network.

- The real motivation behind CAN is the existing networks are not scalable.
- CAN support basic hash table operations on key-value pairs (K,V): insert, search, delete
- CAN is composed of individual nodes and each node stores a chunk (zone) of the hash table
- A hash table is formed as a subset of the (K,V) pairs in the table.
- Each node stores state information about neighbor zones.
- The requests (insert, lookup, or delete) for a key are routed by intermediate nodes using a greedy routing algorithm.
- It do not need any centralized control (completely distributed).
- The small per-node state is independent of the number of nodes in the system (scalable) and also the nodes can route around failures (fault-tolerant).

Properties of CAN

i) Distributed

ii) fault-tolerant

iii) scalable

iv) independent of the naming structure

v) implementable at the application layer

vi) self-organizing and self-healing.

CAN is a logical d-dimensional Cartesian coordinate space organized as a d-torus logical topology, i.e., a virtual overlay d-dimensional mesh with wrap-around.

- A d-torus logical topology is a virtual overlay d-dimensional mesh with wrap-around.
- The entire space is partitioned dynamically among all the nodes present, so that each node i is assigned a disjoint region r(i) of the space.
- As nodes arrive, depart, or fail, the set of participating nodes, as well as the assignment of regions to nodes
- For any object v, its key k(v) is mapped using a deterministic hash function to apoint p in the Cartesian coordinate space.

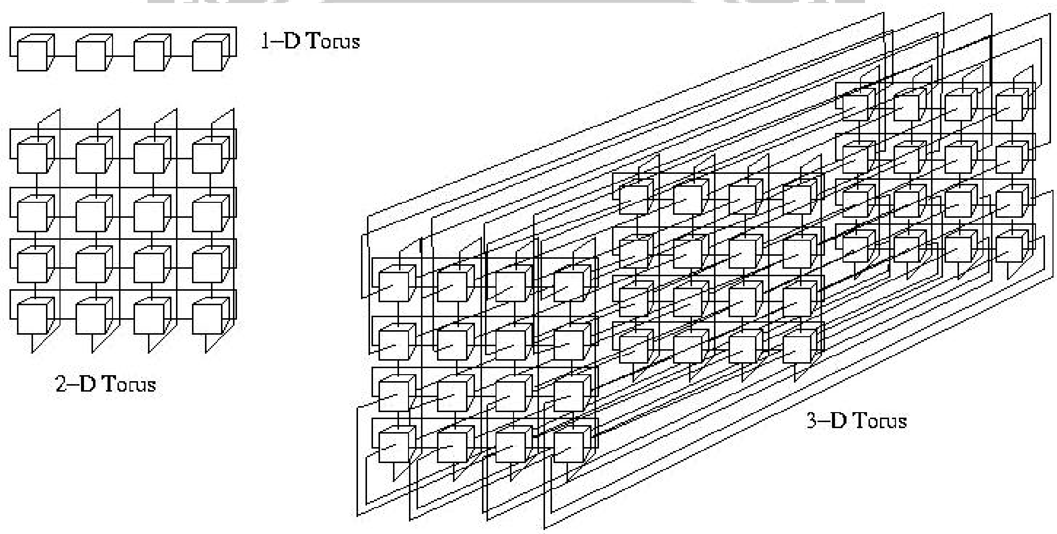

Fig : d-Torus topology

- The (k, v) pair is stored at the node that is presently assigned the region that contains the point p. This means the (k, v) pair is stored at node i if presently the point p corresponding to (k, v) lies in region (r, i).
- To retrieve object v, the same hash function is used to map its key k to the same point p.
- The node that is presently assigned the region that contains p- is accessed to retrieve v.
- The three core components of a CAN design are the following:

1. Setting up the CAN virtual coordinate space, and partitioning it among the nodes as they join the CAN.

2. Routing in the virtual coordinate space to locate the node that is assigned the region containing p.
3. Maintaining the CAN due to node departures and failures.

Initialization of CAN

The following are the steps in CAN initialization:

1. Each CAN is assumed to have a unique DNS name that maps to the IP address of one or a few bootstrap nodes of that CAN.

> ***A bootstrap node is responsible for tracking a partial list of the nodes that it believes are currently participating in the CAN.***

2. To join a CAN, the joiner node queries a bootstrap node via a DNS lookup, and the bootstrap node replies with the IP addresses of some randomly chosen nodes that it believes are participating in the CAN.
3. The joiner chooses a random point p in the coordinate space. The joiner sends a request to one of the nodes in the CAN, of which it learnt in step 2, asking to be assigned a region containing p. The recipient of the request routes the request to the owner old_owner(p) of the region containing p, using the CAN routing algorithm.
4. The old_owner(p) node splits its region in half and assigns one half to the joiner. The region splitting is done using an a priori ordering of all the dimensions, so as to decide which dimension to split along. This also helps to methodically merge regions, if necessary. The (k, v) tuples for which the key k now maps to the zone to be transferred to the joiner, are also transferred to the joiner.
5. The joiner learns the IP addresses of its neighbors from old_owner(p). The neighbors are old_owner(p) and a subset of the neighbors of old_owner(p). The old_owner(p) also updates its set of neighbors. The new joiner as well as old_owner(p) inform their neighbors of the changes to the space allocation, so that they have correct information about their neighborhood and can route correctly. Each node has to send an immediate update of its assigned region, followed by periodic Heartbeat refresh messages, to all

its neighbors.

- When a node joins a CAN, only the neighboring nodes in the coordinate space are required to participate in the joining process.
- The overhead is the order of the number of neighbors, which is O(d) and independent of n, the number of nodes in the CAN.

CAN Routing

- CAN routing uses the straight-line path from the source to the destination in the logical Euclidean space.
- Each node maintains a routing table that tracks its neighbor nodes in the logical coordinate space.
- In d-dimensional space, nodes x and y are neighbors if the coordinate ranges of their regions overlap in d − 1 dimensions, in one dimension.
- All the regions are convex.
- Let the region x be $[[x^1_{min}, x^1_{max}], \ldots [x^a_{min}, x^a]]_{max}$ and the region y be $[[y^1_{min}, y^1_{max}], \ldots [y^d min, y^d max]]$.
- X and y are neighbors if there is some dimension j such that $x^j_{max} = y^i_{min}$ and for all dimensions, $[x^i_{min}, x^i_{max}]]$ and $[y^i_{min}, y^i_{max}]]$ overlap.

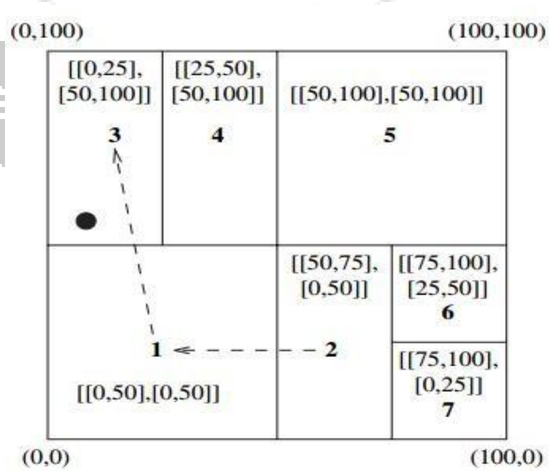

Fig : Two-dimensional CAN space

- The routing table at each node tracks the IP address and the virtual coordinate region of each neighbor.
- To locate value v, its key (k, v) is mapped to a point p- whose coordinates are used in the message header.
- Knowing the neighbors' region coordinates, each node follows simple greedy routing by forwarding the message to that neighbor having coordinates that are closest to the destination's coordinates.
- To implement greedy routing to a destination node x, the present node routes a message to that neighbor among the neighbors k ∈ Neighbors:
- Assuming equal-sized zones in d-dimensional space, the average number of neighbors for a node is O(d).
- The average path length is $(d/4)\ n^{1/d}$.
- The implication on scaling is that each node has about the same number of neighbors and needs to maintain about the same amount of state information, irrespective of the total number of nodes participating in the CAN.
- The CAN structure is superior to that of Chord.
- Unlike in Chord, there are typically many paths for any given source-destination pair.
- This greatly helps for fault-tolerance.
- Average path length in CAN scales as $O(n^{1/d})$ as opposed to log n for Chord.

Maintenance in CAN

- When a node voluntarily departs from CAN, it hands over its region and the associated database of (key, value) tuples to one of its neighbors.
- If the node's region can be merged with that of one of its neighbors to form a valid convex region, then such a neighbor is chosen.
- Otherwise the node's region is handed over to the neighbor whose region has the smallest volume or load – the regions are not merged and the neighbor handles both

zones temporarily until a periodic background region reassignment process runs to integrate the regions and prevent further fragmentation.

- AN requires each node to periodically send a HEARTBEAT update message to each neighbor, giving its assigned region coordinates, the list of its neighbors, and their assigned region coordinates.
- When a node dies, the neighbors suspect its death and initiate a TAKEOVER protocol to decide who will take over the crashed node's region.
- Despite this TAKEOVER protocol, the (key, value) tuples in the crashed node's database remain lost until the primary sources of those tuples refresh the tuples.
- Requiring the primary sources to periodically issue such refreshes also serves the dual purpose of updating stale or dirty objects in the CAN.

TAKEOVER protocol

- When a node suspects that a neighbor has died, it starts a timer in proportion to its region's volume.
- On timeout, it sends a TAKEOVER message, with its region volume piggybacked on the message, to all the neighbors of the suspected failed node.
- When a TAKEOVER message is received, a node cancels its bid to take over the failed node's region if the received TAKEOVER message contains a smaller region volume than that of the recipient's region.
- This protocol thus helps in load balancing by choosing the neighbor whose region volume is the smallest, to take over the failed node's region. As all nodes initiate the TAKEOVER protocol, the node taking over also discovers its neighbors and vice versa.
- In the case of multiple concurrent node failures in one vicinity of the Cartesian space, a more complex protocol using an expanding ring search for the TAKEOVER messages can be used.
- A graceful departure as well as a failure can result in a neighbor holding more than one region if its region cannot be merged with that of the departed or failed node.

- To prevent the resulting fragmentation and restore the 1 → 1node to region assignment, there is a background reassignment algorithm that is run periodically.
- Conceptually, consider a binary tree whose root represents the entire space. An internal node represents a region that existed earlier but is now split into regions represented by its children nodes.
- A leaf represents a currently existing region, and overloading the semantics and the notation, also the node that represents that region.
- When a leaf node x fails or departs, there are two cases:
    1. If its sibling node y is also a leaf, then the regions of x and y are merged and assigned to y. The region corresponding to the parent of x and y becomes a leafand it is assigned to node y.
    2. If the sibling node y is not a leaf, run a depth-first search in the sub tree rootedat y until a pair of sibling leaves (say, z1 and z2) is found. Merge the regions ofz1 and z2, making their parent z a leaf node, assign the merged region to nodez2, and the region of x is assigned to node z1.
- A distributed version of the above depth-first centralized tree traversal can be performed by the neighbors of a departed node.
- The distributed traversal leverages the fact that when a region is split, it is done in accordance to a particular ordering on the dimensions.
- Node i performs its part of the depth first traversal as follows:
    1. Identify the highest ordered dimension dim_a that has the shortest coordinate range $[i^{dima}_{min,}\ i^{dima}_{max}]$. Node i's region was last halved along dimension dim_a.
    2. Identify neighbor j such that j is assigned the region that was split off from i'sregion in the last partition along dimension dim_a. Node j's region i's region along dimension dim_a.
    3. If j's region volume equals i's region volume, the two nodes are siblings

and the regions can be combined. This is the terminating case of the depth first tree search for siblings. Node j is assigned the combined region, and node i takes over the region of the departed node x. This take over by node i is done by returning the recursive search request to the originator node, and communicating i's identity on the replies.

4. Otherwise, j's region volume must be smaller than i's region volume. Node i forwards a recursive depth-first search request to j.

CAN Optimizations

The following are the design techniques to improve the performance of factors:

- **Multiple dimensions:** As the path length is $O(d \cdot n^{1/d})$, increasing the number of dimensions decreases the path length and increases routing fault tolerance at the expense of larger state space per node.
- **Multiple realities:** A coordinate space is termed as a reality. The use of multiple independent realities assigns to each node a different region in each different reality. This implies that in each reality, the same node will store different (k, v) tuples belonging to the region assigned to it in that reality, and will also have a different neighbor set. The data contents (k, v) get replicated in each reality, leading to higher data availability. The multiple copies of each (k, v) tuple, one in each reality, offer a choice – the closest copy can be accessed. Routing fault tolerance improves because each reality offers a set of different paths to the same (k, v) tuple. All these contribute to more storage.
- **Delay latency:** The delay latency on each of the candidate logical links can also be used in making the routing decision.
- **Overloading coordinate regions:** Each region can be shared by multiple nodes, up to some upper limit. This reduces path length and path latency. The fault tolerance improves because a region becomes empty only if all the nodes assigned to it depart or fail concurrently. The per-hop latency decreases because a node can select the closest node from the neighboring region to forward a message towards the destination. This demands many of the aspects of the basic CAN protocol need to be reengineered to

accommodate overloading of coordinate regions.

- **Multiple hash functions:** The use of multiple hash functions maps each key to different points in the coordinate space. This replicates each (k, v) pair for each hash function used. The effect is similar to that of using multiple realities.

- **Topologically sensitive overlay:** The CAN overlay has no correlation to the physical proximity or to the IP addresses of domains. Logical neighbors in the overlay may be geographically far apart, and logically distant nodes may be physical neighbors. By constructing an overlay that accounts for physical proximity in determining logical neighbors, the average query latency can be significantly reduced.

CAN Complexity

- The time overhead for a new joiner is O(d) for updating the new neighbors in the CAN, and O(d/4 log(n)) for routing to the appropriate location in the coordinate space.
- The time overhead and the overhead in terms of the number of messages for a node departure is $O(d^2)$, because the TAKEOVER protocol uses a message exchange between each pair of neighbors of the departed node.

.