

MIDAS: Multi-Attribute Indexing for Distributed Architecture Systems

George Tsatsanifos (NTUA, R.C. "Athena")
Dimitris Sacharidis (R.C. "Athena")
Timos Sellis (NTUA, R.C. "Athena")

August 22, 2011

Outline

- 1 Introduction
- 2 Related Work
 - CAN
 - VBI-Tree
 - P-Grid
- 3 MIDAS
 - Structure
 - Overlay Operations
 - Fault-Tolerance
 - Lookup Queries
 - Range Queries
- 4 Experiments
 - Simulations
 - Results
- 5 Conclusions

Peer-to-Peer Networks

Structured vs. Unstructured

Unstructured P2P

- Arbitrary links
- Flooding
- Random walks
- Query resolution?

Structured P2P

- Employment of a globally consistent protocol
- Routing through network structure

Query Types

Contemporary DHTs support/approximate

- Exact Queries (lookups).
- Range Queries of low dimensionality.
- Nearest Neighbors Queries (KNN).
- Aggregation (min, max, avg, sum).

CAN

Index Structure

- A virtual d -dimensional Cartesian coordinate space on a d -torus.
- Two nodes are neighbors if their coordinates spans overlap along $d - 1$ dimensions and abut along one.
- Neighbors serve as a coordinate routing table, of cardinality $\Theta(d)$, enabling routing between arbitrary points.
- Node forwards a message through its neighbor with coordinates closest to the destination.
- Routing path length in $O(\sqrt[d]{n})$ hops.

CAN

Overlay Operations

Join

- 1 Find a node already in the CAN.
- 2 Find a node whose zone will be split.
- 3 Neighbors of the split zone must be notified.

Departure

If no neighbor's zone can be merged to a valid single zone, then it is handed to the neighbor with the smallest zone.

Failure

One of the failed node's neighbors takes over the zone. Data held by the failed node will be lost until the state is refreshed by their holders.

The VBI-Tree

Index Structure

- VBI supports a variety of indexing methods such as the R-Tree, X-Tree, SSTree, and M-Tree.
- **However, limited to binary tree structures.**
- Each node maintains links to its parent, its children, its adjacent nodes and its sideways routing tables.
- The basic idea is to assign a region of the attribute space to each data node.
- Each internal node has associated a region that covers all regions managed by its children.

The VBI-Tree

Overlay Operations

Node Joins

- Cost of finding a node to join: $O(\log N)$
- When a node accepts a new node as its child
 - Split half of its content (its range of values) to its new child.
 - Update adjacent links of itself and its new child
 - Notify both its neighbor nodes and its new child's neighbor nodes to update their knowledge
 - Cost: $6 \log N$

The VBI-Tree

Node Departures

- Leaf nodes with no neighbors having children can leave the network
 - Transfer content to the parent node, and update correspondence adjacent link.
 - Notify neighbor nodes and parents neighbor nodes to update their knowledge.
 - Cost: $4 \log N$
- Leaf nodes with a neighbor node having children, need to find a leaf node to replace them from the children of that neighbor node.
- Intermediate nodes need to find a leaf node to replace them from their adjacent nodes.

Fault Tolerance

Node failures

- Nodes discovering failure of a node report to that nodes parent.
- The failures parent node will take responsibility for finding a leaf node to replace if necessary.
- Routing information of the failure node can be recovered by contacting its neighbor nodes via routing information of its parent.

Fault tolerance: failure node can be passed by two ways

- Through routing tables - horizontal axis
- Through parent-child and adjacent links - vertical axis

P-Grid

Index Structure

- Consider a binary trie whose leaves correspond to actual peers.
- Each peer is identified by a unique binary identifier which corresponds to the route from the root to the leaf standing for the peer.
- Each peer is responsible for all keys which have a prefix identical to peer's identifier.
- A peer maintains routing information for each bit of its identifier, with at least one other peer participating in the network with that specific bit inverted.

MIDAS Structure

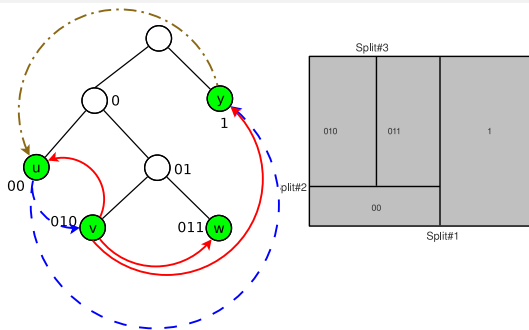
Index Structure

MIDAS under the scope...

- Virtual distributed kd-tree.
- Part of the virtual tree hierarchy is represented by *split history* (node's path) and *split points* in each overlay participant.
- Their combination defines the position of a coordinated zone in space (range partitioned network).
- For each node in its path from the root, a peer knows another peer from the subtree it does not belong.
- Tuples stored into the leaf nodes of the appropriate responsibility area.
- *No global knowledge!*

Structure

Example



| Peer | <i>link entries</i> | | |
|----------|---------------------|----------|----------|
| $u(00)$ | $y(1)$ | $v(010)$ | |
| $v(010)$ | $y(1)$ | $u(00)$ | $w(011)$ |
| $w(011)$ | $y(1)$ | $u(00)$ | $v(010)$ |
| $y(1)$ | $u(00)$ | | |

Fundamental Operations

Elementary Functionality

Node Insertion

A new node is created by expanding the tree, through splitting a leaf node along its most spread dimension and expand the kd-tree.

Node Removal

A node u is removed when merged with any neighboring peer v , where u, v overlap along $d - 1$ dimensions and abut along one ($u.depth=v.depth$ in the tree).

Fault-Tolerance

Node Failures and Robustness

Two cases for peer u that traces the failure of a known peer w .

- 1 If failed peer w is the last link of u 's routing table, then u takes over w 's area of responsibility.
- 2 Otherwise, peer u bypasses w by issuing a lookup query for a random point in the area designated by the j -th split point, and replaces the failed link with the owner.

Lookup Queries

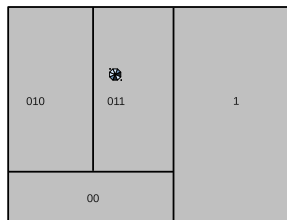
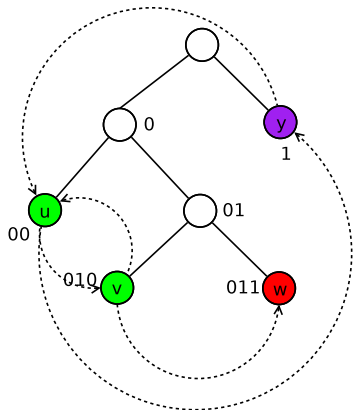
Exact Search

- 1 Issue/Receive a lookup query.
- 2 If it can be processed locally, then answer it.
- 3 Else traverse the local virtual tree hierarchy for the most relevant node known and forward the query.
- 4 This procedure is repeated recursively $O(\log n)$ hops until the request reaches the responsible node.

Lookup Queries

An exact search example

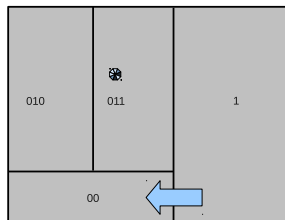
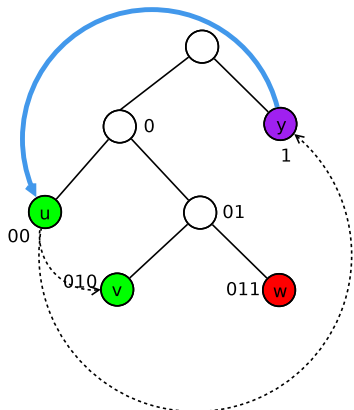
A lookup query is issued by node $y(1)$ for the query point \vec{q} (diamond).



Lookup Queries

An exact search example

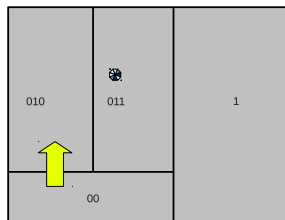
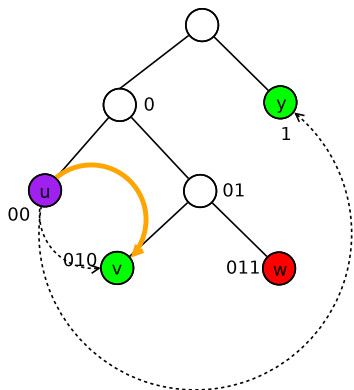
Since node $y(1)$ cannot answer the query locally, it is forwarded through the link corresponding to the node on the left of the first split-point.



Lookup Queries

An exact search example

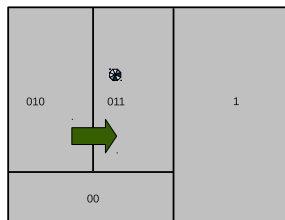
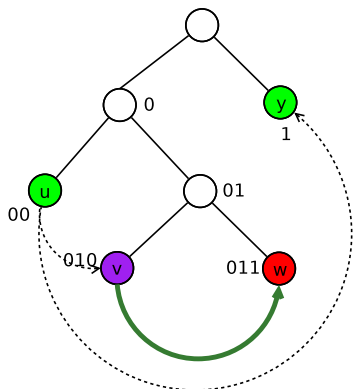
But node $u(00)$ can neither retrieve the key locally, and therefore, will recursively forward the request to the most relevant node it knows, the node that corresponds above the second split-point.



Lookup Queries

An exact search example

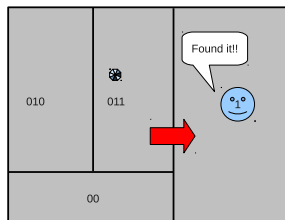
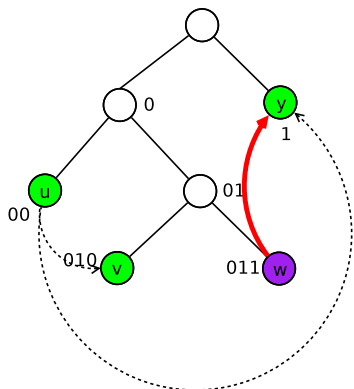
The request reaches node $w(011)$ that is responsible for the queried key.



Lookup Queries

An exact search example

Eventually, node $w(011)$ returns the $(key, value)$ pair to the issuer node.



Range Queries

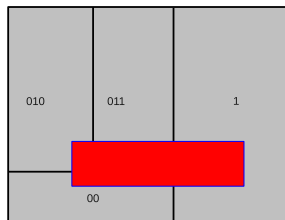
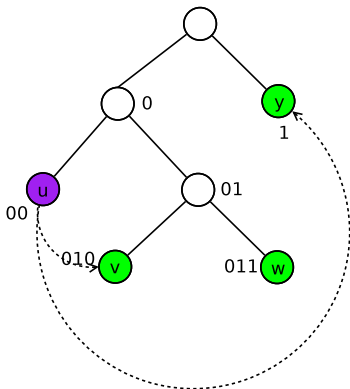
Orthogonal Search Method

- 1 Issue/Receive a range query.
- 2 If the request coverage area is not fully enclosed, then forward to known relevant nodes a part of the query (defined by split-points).
- 3 If there is any overlapping between local responsibility area and request coverage area, then answer that part.
- 4 This procedure is repeated recursively in $O(\log n)$ hops until the query is satisfied.

Range Queries

An orthogonal search example

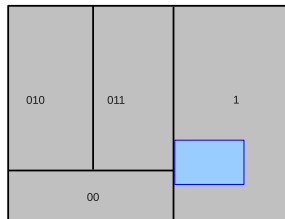
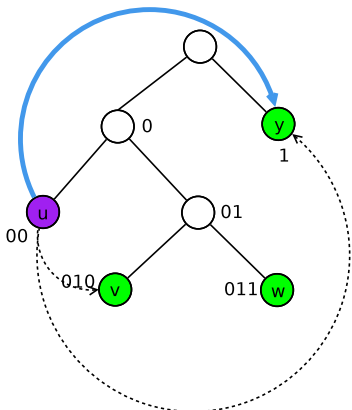
A range query is issued by node $u(00)$ for the red area.



Range Queries

An orthogonal search example

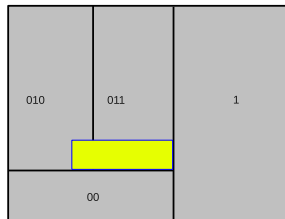
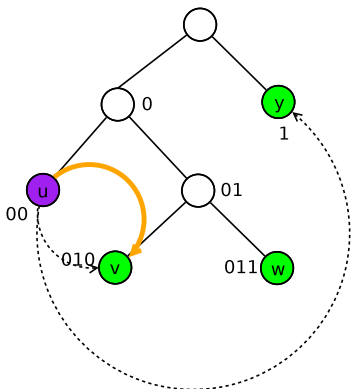
The query is partially answered by node $y(1)$ (one hop) - query fragmented along the first split-point.



Range Queries

An orthogonal search example

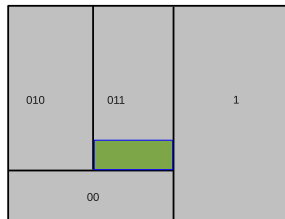
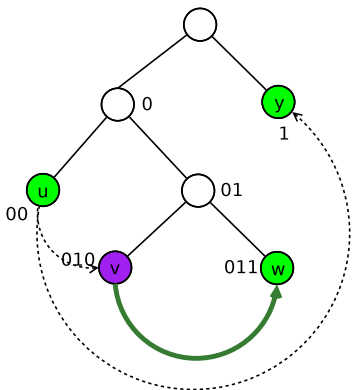
The query is partially answered by node $v(010)$ (one hop) - remaining query after fragmented along the second split-point.



Range Queries

An orthogonal search example

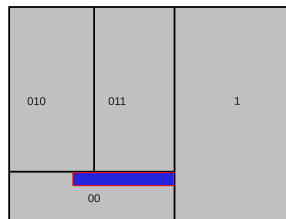
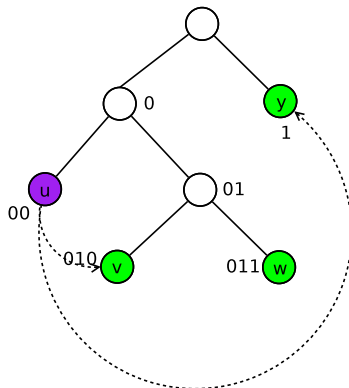
Also partially answered by node $w(011)$ (two hops away) - subquery fragmented along the first, second and third split-points



Range Queries

An orthogonal search example

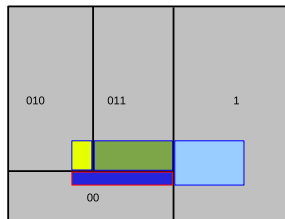
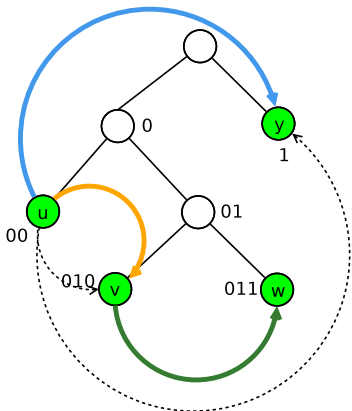
The remainder of the query is partially processed locally by node $u(00)$.



Range Queries

An orthogonal search example

A range query issued by $u(00)$ and partially answered by $y(1)$ and $u(010)$ (one hop), and $w(011)$ (two hops).



Experimental Evaluation

Setting

Simulations

- Our experiments simulate a dynamic environment.
- They consist of two stages, a *growing* and a *shrinking* stage.
- *Real spatial* and *synthetic* high-dimensional datasets.
- We initiate an overlay of 1K peers, increasing to 70K peers ...followed by the reverse procedure.
- Datasets consist of 1M keys.
- Querysets consist of 50K queries.
- Each range query evaluates 50 tuples.

Lookups Evaluation

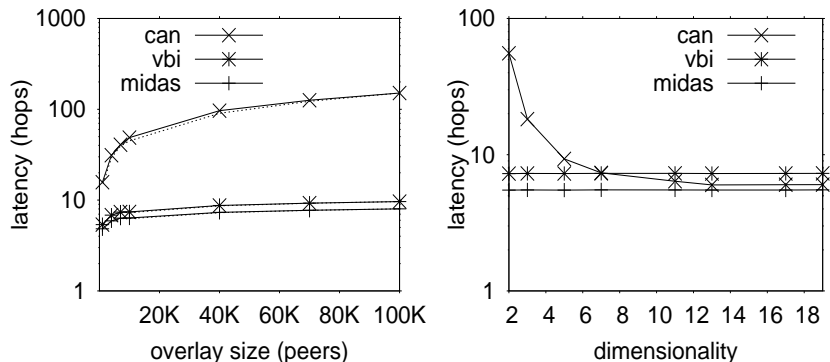


Figure: Latency for lookup queries for MIDAS, VBI, CAN a peer maintains.

Range Queries Evaluation

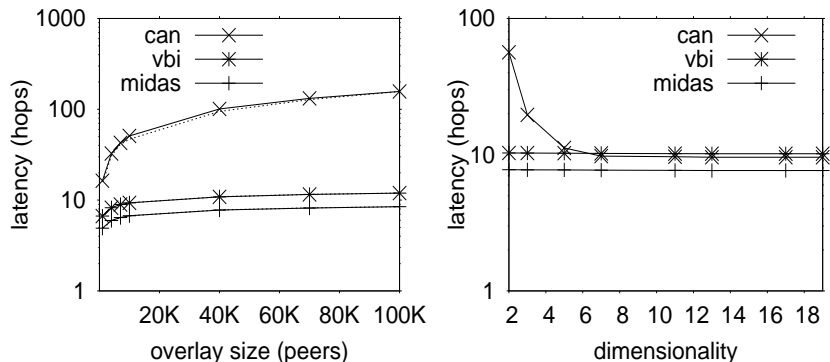


Figure: Latency for range queries for MIDAS, VBI, CAN a peer maintains.

Conclusions

- Pure multi-dimensional paradigm.
- Enhanced scalability and performance.
- Requests are satisfied in $O(\log n)$ hops.
- Increased dimensionality has no effect on queries.
- Skewness affects latency and data load fairness only slightly.

Questions?

