Distributed Index for Multi-dimensional (DIM) Data in Sensor Networks

Xin Li, Young Jin Kim, Ramesh Govindan, Wei Hong
USC/ENL, Intel Research Berkeley
Outline

• Introduction
  – Motivation, overview, and related work

• Algorithms
  – Zones, data insertion, query propagation, robustness

• Analysis and simulation results
  – Cost analysis
  – ns-2 simulation

• Implementation
Motivation

• Treat a sensor network as a database and use \textit{in-network storage}.

• Provide an efficient solution for multi-dimensional range queries.
  – e.g. List all events whose temperature lies between 70 and 80 and whose light levels are between 10 and 15.

• Useful for searching and correlating events of interests with multiple attributes.
  – Drill-down searching, trigger and action, …
DIM Overview

- **Functionality**
  - Efficient range query for multidimensional data.

- **Approaches**
  - Divide sensor field into bins.
  - Locality preserving mapping from m-d space to geographic locations.
  - Use geographic routing such as GPSR.

- **Assumptions**
  - Nodes know their locations and network boundary
  - No node mobility

\[ E_2 = <0.6, 0.7> \]
\[ E_1 = <0.7, 0.8> \]
\[ Q_1 = <0.5, 0.7, 0.5, 1> \]
Related Work

- **Database Indices**
  - k-d trees, other indices
  - Mostly centralized

- **Distributed Hash Tables (DHT)**
  - CAN, Chord, …
  - Logical overlay independent of network topology

- **Sensor Network Database**
  - TinyDB, Semantic Routing Trees (SRT)
  - For query dissemination, not for storage

- **Data-Centric Storage**
  - GHT for exact match queries
  - DIMENSIONS, DIFS
Outline

• Introduction
  – Motivation, overview, and related work
• Algorithms
  – Zones, data insertion, query propagation, robustness
• Analysis and simulation results
  – Cost analysis
  – $ns$-2 simulation
• Implementation
Building Zones

- Divide network into zones.
- Each node mapped to one zone.
- Encode zones based on division.
- Each zone has a unique code.
- Map m-d space to zones.
- Zones organized into a virtual binary tree.

$L: \text{Light}, T: \text{Temperature}$
Virtual Binary Tree
Challenges in Zone Building

- Handling empty zones
  - DIM assigns each zone an owner
  - Owner is the closest node on the tree
  - Robustness support uses new owner as the local replication.

- Resolving code conflict
  - Lazy resolution
  - Driven by event or query messages
  - Resort to GPSR
GPSR

- Greedy Perimeter Stateless Routing (B. Karp, et al)
  - Greedy mode
  - Perimeter mode when Greedy failed.
- DIM uses GPSR in
  - Packet Delivery
  - Owner Discovery
    » Find the owner of an empty zone
    » Resolve conflicting ownership claims
    » Look for backup-node (Robustness)
Data Insertion

- Encode events
- Compute geographic destination
- Hand to GPSR
- Intermediate nodes can refine the destination estimation

$L: \text{Light}, T: \text{Temperature}$
Query

- Split a large query into smaller subqueries.
- Encode each subquery.
- Process subqueries separately, resolving locally or forwarding to other nodes based on their codes.

$L$: Light, $T$: Temperature
Robustness

• Data Robustness
  – Local Replication
    » Back-up node = the new owner of its zone if a node failed.
    » Good for random node failures — at least one copy is accessible.
  – Mirror Replication
    » Switching the zone codes in the original DIM: 110 → 001.
    » Resilient to concurrent failures of geographically contiguous nodes.
    » Double both insertion and query costs.

• Query Robustness
  – Resilient to packet loss
  – Selectively reissue part of a query to cover silent zones
Outline

• Introduction
  – Motivation, overview, and related work

• Algorithms
  – Zones, data insertion, query propagation, robustness

• Analysis and simulation results
  – Cost analysis
  – $ns$-2 simulation

• Implementation
**DIM Analysis**

- Average insertion cost \( O(\sqrt{N}) \), for a \( N \) node dense network.
- Average query resolution cost \( \int_{1}^{N} xf(x)dx \), for query size \( x \) and probability density function \( f(x) \).

<table>
<thead>
<tr>
<th>Query size distribution</th>
<th>Average query cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>Uniform with bound ( B )</td>
<td>( O(B) )</td>
</tr>
<tr>
<td>Algebraic: ( f(x) \propto e^{-cx} )</td>
<td>( O(\sqrt{N}) )</td>
</tr>
<tr>
<td>Exponential: ( f(x) \propto x^{-k} )</td>
<td>( O(\sqrt{N}) )</td>
</tr>
</tbody>
</table>

When most queries are small, the average query cost of DIM can achieve \( O(\sqrt{N}) \).
**ns-2 Simulation**

- **Scenarios**
  - Nodes are randomly placed.
  - Three data distributions: Uniform, Normal, Skewed.
  - Four query size distributions
    » Uniform, Uniform with bound $B$, Power Law, Exponential.
    » Queries are randomly placed in the data space.

- **Comparison with**
  - Flooding
    » Data stored where they are generated
    » Queries are flooded to the entire network
  - GHT-R
    » For small range queries, issue an exact match query for each countable value in the range.
Comparison

DIM outperforms flooding and GHT-R in small size queries.
Local Replication Performance

With local replication scheme, DIM has a graceful degradation under random node failures.
Outline

• Introduction
  – Motivation, overview, and related work

• Algorithms
  – Zones, data insertion, query propagation, robustness

• Analysis and simulation results
  – Cost analysis
  – ns-2 simulation

• Implementation
Architecture on Motes

- Use TinyDB Schema
- Tuple Storage on Flash/RAM
  - Insertion, deletion, search
- GPSR implemented with priority-based hop-by-hop reliability
  - Favor query and reply messages different over other messages.
- DIM’s components
  - Zone, Query, Insertion,
  - Dispatcher
Come to See Our Demo!

![Image of Demo GUI windows showing data tables and attributes]

Select attributes and their ranges for query #0

Results for query #1

<table>
<thead>
<tr>
<th>Sender</th>
<th>Detector</th>
<th>Timehi</th>
<th>Timelo</th>
<th>light</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1387904</td>
<td>1003</td>
<td>465</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1385408</td>
<td>553</td>
<td>102</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0</td>
<td>1392224</td>
<td>983</td>
<td>299</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1363808</td>
<td>430</td>
<td>347</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1411616</td>
<td>625</td>
<td>363</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1365408</td>
<td>401</td>
<td>135</td>
</tr>
</tbody>
</table>
Conclusion

• DIM provides an efficient solution for multi-dimensional range queries in sensor networks
  – Divide the sensor field into zones
  – Map data and queries into zones using a locality preserving hashing
  – Use geographic routing such as GPSR

• Future work
  – Software release
  – Integration with TinyDB
  – Rebalancing
    » Storage hot-spot handling
    » Rebuild DIM when data distributions known