2009 15th International Conference on Parallel and Distributed Systems

A Decentralized Storage Scheme for Multi-Dimensional Range Queries over Sensor Networks

Lei Xie, Lijun Chen, Daoxu Chen, Li Xie

State Key Laboratory of Novel Software Technology
Department of Computer Science, Nanjing University, Nanjing, China
Email: xielei@dislab.nju.edu.cn, {chenlj,cdx,xieli}@nju.edu.cn

***Abstract*—This paper presents the design of a decentralized storage scheme to support multi-dimensional range queries over sensor networks. We build a distributed k-d tree based index structure over sensor network, so as to efficiently map high dimensional event data to a two-dimensional space of sensors while preserving the proximity of events. We propose a dynamic programming based methodology to control the granularity of the index tree in an optimized approach, and an optimized routing scheme for range query processing to achieve best energy efficiency. The simulation results demonstrate the efficiency of the design.**

I. Introduction

In most sensor network applications, data or events are continuously collected, forwarded and stored in sink nodes for further usage. Recently a *Data Centric Storage* (DCS) scheme is proposed to store event data in specific sensor nodes according to the "event type", leveraging the "in-network storage" capability of sensor networks. The "event type" refers to certain pre-defined constellations of attribute values, however, earlier DCS schemes like GHT [1] conventionally deal with event types with unique attribute and only support "exact match" point queries. As sensor applications are evolving, it is mandatory for these event data to be represented by several different attributes, such as temperature, humidity, light, and barometric pressure, which may be referred to as multi-dimensional events. One natural way to query for events of interest is to use multi-dimensional range queries over these attributes. For example, users may impose queries like "List all events that have temperatures between 50.F and 60.F, and light levels between 10 and 20". However conventional DCS schemes are inadequate for managing and processing these multi-dimensional events, because processing a query over multi-dimensional events is much more complex than processing a query on single-dimensional events. We need to figure out how to efficiently map these high dimensional events to a two-dimensional space of sensors while preserving the proximity of events.

Therefore an efficient, decentralized storage framework is essential to facilitate multi-dimensional queries over sensor networks. In this paper we propose a framework which can efficiently handle the event insertions and range query processing in an optimized approach. Contributions of this paper are listed as follows:

1. We construct the k-d tree based index structure according to event and range query distributions at sensor data level instead of the two-dimensional sensor locations, thus we can well utilize data level information to facilitate the event insertion and query processing.

2. We reduce the problem of optimal routing for sub-queries to the *Euclidean Steiner tree* problem, which is NP-hard, and propose corresponding approximation algorithm to solve it.

The rest of the paper is organized as follows. Section II reviews the related work. Section III provides preliminaries upon which the design theory can be built. The details of the proposed design methodology are given in Section IV. In Section V, we present the multi-dimensional event storage and query processing mechanism. Section VI presents the simulation results. Finally, we summarize the paper in Section VII.

II. Related Works

Data centric storage schemes [2] mainly utilize a mapping function such as geographic hash function to map event information to specified geographic points, and leverage routing algorithms like GPSR [1] to route messages to corresponding storage nodes, these research works include GHT [1], DIMENSIONs [3] and DIFS [4]. GHT is the first work which utilizes a geographic hash table as the mapping function. Based on the primitives in GHT, DIMENSIONs is proposed to support multi-resolution storage and query processing, which allows drill-down search for features within sensor network, while DIFS allows range queries on a single key in addition to other operations. And these above storage schemes are mainly suitable for processing one-dimensional events.

The basic problem that this paper addresses - multidimensional range queries - is typically solved in database systems using indexing techniques. Among these techniques k-d tree[5] as one of the well-known multi-dimensional binary search trees, is proposed to support associative searching works over multi-dimensional data attributes. DIM[6] draws its inspiration

1521-9097/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPADS.2009.24

from the k-d tree index structure, and embeds the k-d tree into sensor networks in a decentralized approach. Furthermore, KDDCS [7] presents a load-balanced storage scheme over sensor networks based on k-d tree for multi-dimensional range queries. POOL[8] addresses some weakness of DIM such as hot spot and scalability problems, and further proposes a storage scheme to group index nodes together as a data pool to preserve the neighborhood property of nearby multi-dimensional data, however, their group mechanism are only based on the greatest and the second greatest attribute values, which has not been proved to be efficient enough for high dimensional data processing.

III. PRELIMINARIES

A. Motivation

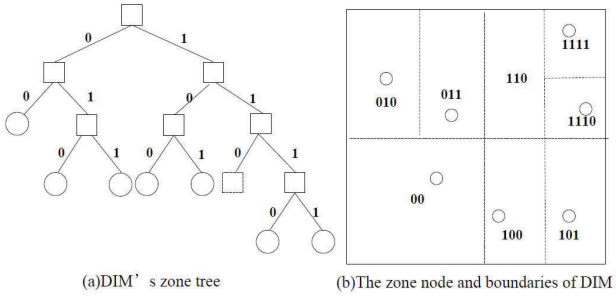


Fig. 1. Example of DIM (Distributed Index for Multi-dimensional data

It is essential to construct an index structure to map the multi-dimensional event data to two-dimensional sensors. Obviously a centralized index for multi-dimensional range queries is not feasible for energy-efficiency reasons of sensor networks. DIM [6] is the first work to support multi-dimensional range queries in sensor networks, it utilizes a classical database index, the k-d tree [5], and embeds such an index in a sensor network. DIM divides sensor nodes into separate zones and each zone contains one exact sensor node as the index node. Fig.1(a) depicts an example of DIM's k-d tree for zones and Fig.1(b) shows the corresponding zones over the sensor network. Each intermediate node over this k-d tree based index structure denotes the split parameters for further branching, which include the specified attribute $A_i (1 \leq i \leq m)$ from the attribute list $\langle A_1, ..., A_m \rangle$, and associative splitting ranges for A_i. Each leaf node represents the event data within specified multi-dimensional ranges.

We introduce the principle of k-d tree to process multi-dimensional attribute data with the following example: For attributes $\langle A_1, ..., A_m \rangle$, assume that the maximum depth of the k-d tree is k, k is a multiple of m, and this value of k is known to every node. DIM assigns a k bit zone code to an event as follows. For i between 1 and m, if $A_i < 0.5$, the i-th bit of the zone code is assigned 0, else 1. For i between $m+1$ and $2m$, if $A_{i-m} < 0.25$ or $A_{i-m} \in [0.5, 0.75)$, the i-th bit of the zone is assigned 0, else 1, because the next level divisions are at 0.25 and 0.75 which divide the ranges to [0, 0.25), [0.25, 0.5), [0.5, 0.75), and [0.75, 1). They repeat this procedure until it reaches the corresponding leaf node. As an example, consider event $E = (0.3, 0.8)$, for this event, the zone code over the aforementioned example k-d tree is code $(Z_E) = 011$. When answering a range query, DIM finds all zones whose value ranges overlap with the query range and sends the query to index nodes of those zones. For example, consider a range query $\langle [0.3, 0.7], [0.5, 0.6], [0.4, 0.8], [0.7, 0.9] \rangle$, the storage regions of zone code 010, 011, 110, 1111 will be involved.

Building upon this, DIM achieves energy efficiency by forcing events whose attribute values are "close" to be stored at the same or nearby nodes. However, some performance issues still remain to be challenged for the traditional DIM: First, DIM constructs the k-d tree structure only based on the two-dimensional sensor locations, without leveraging the estimated distributions of various data attributes and range queries, thus it may cause inefficiency for non-uniform event insertions and corresponding range queries, which are actually conventional for sensor network applications. Second, DIM assigns a unique zone for each sensor node, hence as the network size increases, the zones have to be further split into smaller zones, more zones of sensors are inevitably involved in the query processing, which cannot be energy efficient for range query processing. Therefore how to control the granularity of the k-d tree based index structure is crucial for energy efficiency issue. Third, DIM has not considered an optimized approach to route those split sub-queries to specified storage areas, since the query diffusion cost dominates the overall energy consumption for multi-dimensional range query processing, it is of utmost importance to devise an optimized routing scheme for query forwarding.

Therefore in this paper we try to address the problems raised above and we propose a framework which can efficiently handle the event insertions and range query processing in an optimized approach. Our storage framework is also based on the k-d tree and we mainly focus on achieving the optimized granularity of k-d tree structure over sensor networks for best energy efficiency, and we further investigate into the optimized scheme for range query forwarding which dominates the overall energy consumption.

B. Modelling Event Data Distribution

Consider there exist k attributes $A_1, A_2, ..., A_k$ for sensor data, for each attribute A_i, we have minimum value $\min(V_i)$ and maximum value $\max(V_i)$, we normalize attribute A_i's value into the region $[0, 1]$ by calculating:

$$V_i^{'} = \frac{V_i - \min(V_i)}{\max(V_i) - \min(V_i)} \tag{1}$$

And we represent the data distribution over attribute A_i as $p_i(x)$, which is exactly *a probability density function* (*pdf*) over the region $x \in [0, 1]$, thus we have

$$\int_0^1 p_i(x)dx = 1 \tag{2}$$

we can also define a discrete form of $p_i(x)$ as follows, we equally divide the region $[0, 1]$ into m sub-regions, and leverage the histogram method to calculate the probability $p_{i,j}$,$(j = 1, ..., m)$ and we have:

$$\sum_{j=1}^{m} p_{i,j} = 1 \tag{3}$$

Hence consider an index node over the k-d tree with multi-dimensional range $[(\alpha_1, \beta_1), (\alpha_2, \beta_2), \cdots, (\alpha_k, \beta_k)]$, where α_i and β_i respectively denote the lower bound and upper bound of specified range of attribute A_i, we have $0 \leq \alpha_i < \beta_i \leq 1$, assume the above multi-dimensional range is associated with a two-dimensional spacial region S_j, therefore we can calculate the probability that a random event data falls into this region S_j as:

$$Prob_E(S_j) = \prod_{i=1}^{k} \int_{\alpha_i}^{\beta_i} p_i(x)dx \tag{4}$$

Assume the overall event frequency is f_e, thus the frequency for which the specific storage region S_j is accessed is $f_e \cdot Prob_E(S_j)$.

C. Modelling Range Query Distribution

Unlike event distributions over attributes, which are mutually exclusive over each attribute A_i, the range query distributions over attributes are overlapped because each query is corresponding to a range over the attribute dimensions. Users can specify an k dimensional range query in the form like $\langle [L_1, U_1], [L_2, U_2], ..., [L_k, U_k] \rangle$ over k attributes, where $0 \leq L_i \leq U_i \leq 1, (1 \leq i \leq k)$, for partial range query which does not include attribute A_i, we can set $L_i = 0, U_i = 1$. Thus we utilize another method to depict the distributions: Given a training set of queries Q, we have $Q = \{Q_1, Q_2, ..., Q_N\}$, where query Q_j has a query range over each attribute A_i, which is (L_i, U_i), hence we define an indicator variable B_i for a specified range (α_i, β_i) of attribute A_i as follows:

$$B_i = \begin{cases} 1 & \text{if } (\alpha_i, \beta_i) \cap (L_i, U_i) \neq \phi \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

And we further calculate the query frequency f_i for the specified range (α_i, β_i) over attribute A_i:

$$f_i = \sum_{\forall Q_j \in Q} B_i \tag{6}$$

Thus the specified attribute range will be queried with probability $p_i = f_i/N$.

Assume query distributions over various attributes are mutually independent, given a multi-dimensional data region which is associated with a two-dimensional spacial region S_j, we can finally get the probability for the specified storage region S_j to be queried as:

$$Prob_Q(S_j) = \prod_{i=1}^{k} p_i \tag{7}$$

Assume the overall query frequency is f_q, thus the frequency for which the specified storage region S_j is queried is $f_q \cdot Prob_Q(S_j)$.

IV. Design of Decentralized Storage Scheme for Multi-Dimensional Range Queries

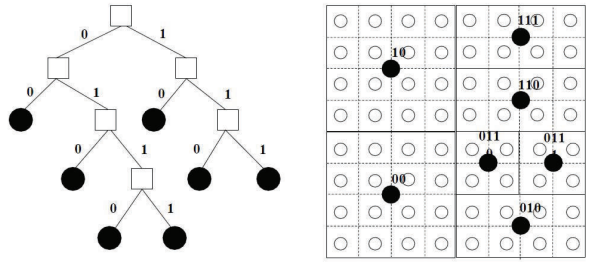


Fig. 2. Decentralized Storage Scheme for Multi-Dimensional Range Queries

We visualize the deployment field of a sensor network as a number of $n \times n$ equal sized grid cells. As DIM mainly applies for sparse sensor network where some zones may not have any sensor node, in this paper we consider the situation where the node density is high enough for each cell to have at least one storage node candidates, these cells are nominated as the storage units of the DCS scheme, at the center of each cell one exact sensor node is selected as the storage node.

A. Indexing Storages over Cells

We construct the index structure based on k-d tree over the cell-based storage deployment area, on one hand, each index node over the index structure represents a multi-dimensional range for the sensor data, on the other hand, it also associates with a specified spacial region over the two-dimensional deployment area. The definition of the index structure is consistent with DIM, except that we utilize the cell as the storage unit instead of sensor node, each index node corresponds to a spacial region which covers one or more cells. Fig.2 shows an example of our decentralized storage scheme to support multi-dimensional range queries. Fig.2(a) depicts the k-d tree based index structure and Fig.2(b) represents the corresponding zone code and boundaries. The black nodes denote the storage regions specified by the k-d tree and those cell(s) contained in the corresponding regions become(s) the replicate storage cells, which means each cell will store the same copy of the multi-dimensional ranged data specified by the index node over the k-d tree.

B. Controlling Granularity for k-d Tree based Index Structure

Now we investigate into the k-d tree based index structure, as already mentioned in Section I, we note that controlling the granularity of the index nodes is very crucial for energy efficiency of multi-dimensional event insertion and range query diffusion. If we split these sensor data into very refined zones according to multi-dimensional data attributes, thus to resolve a conventional range query, we have to split it into many sub-range queries for various zones, apparently it cannot be energy efficient due to the large number of routing cost for

query forwarding from query node to these various storage nodes. Based on the above observation, keeping these ranges of data to be hosted in a relatively small number of zones may be more efficient, but obviously the extreme situation to keep all ranges of data in one unique zone (with replicated storage scheme) is also not energy efficient, because although the query forwarding cost can be effectively reduced, large amount of energy will be drained for routing the same copy of sensor data to all replicate storages. Based on the above analysis, we must find an optimized storage strategy to control the granularity for the index structure as a tradeoff between the two extremum situations to achieve best energy efficiency.

Therefore to construct an optimized k-d tree based index structure over the sensor network, if we build the k-d tree from the root, then at each splitting step of the k-d tree, we have two optional strategies: one strategy is to conduct further splitting for the specified zones, another strategy is to stop further splitting and thus apply the replicate storage scheme to all cells covered by the specified zones. The most crucial thing to construct the index structure is how to decide the optimized strategy at each splitting step along the k-d tree, so as to achieve the best energy efficiency.

To solve the above optimization problem, we start by first analyzing the energy cost for event insertion and query processing over a specified storage region. In this paper we utilize the *Euclidean Distance* as the metric of energy consumption, which means the routing cost between two nodes is represented by the Euclidean distance between them. For insertion of event, we leverage the double ruling scheme[9][10] to distribute the event information to all relevant replicate storage nodes in the storage region. For one specified storage region, the event node first routes event message to the nearest storage node inside the region, and further forwards it both horizontally and vertically to all storage nodes covered by the storage region. Thus the expected energy cost for event distribution over the specified storage region S_i is:

$$E_{event}(S_i) = D_{event}(S_i) + d \cdot (N(S_i) - 1) \tag{8}$$

where $D_{event}(S_i)$ denotes the expected distance from an arbitrary event node to the nearest storage node in storage region S_i, and d is the cell size, $N(S_i)$ denotes the number of storage nodes inside S_i, as each storage node inside S_i receives exactly one message except the first accessed storage node, thus the diffusion cost inside S_i is $d \cdot (N(S_i) - 1)$. Assume the root of the k-d tree is at level 0, and the k-d tree indexes the cell unit at level h, (apparently h should be an even number), thus at level i $(0 \leq i \leq h)$, we have:

$$N(S_i) = 2^{h-i} \tag{9}$$

And for query diffusion, we only need to reach one of those specified replicate storage nodes, an intuitive strategy is for the query node to find the nearest storage node in the storage region S_i, later we will introduce an optimized routing scheme for query diffusion, thus the expected energy cost for query diffusion over the specified storage region S_i is:

$$E_{query}(S_i) = D_{query}(S_i) \tag{10}$$

where $D_{query}(S_i)$ denotes the expected distance from an arbitrary query node to the nearest storage node in the storage region. And we can simply deduce that $D_{query}(S_i)$ equals to $D_{event}(S_i)$.

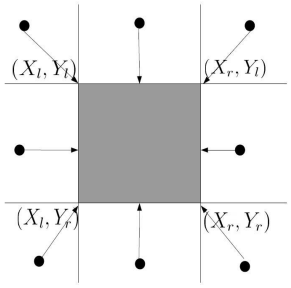


Fig. 3. Estimate the value for $D_{query}(S_i)$ or $D_{event}(S_i)$

We estimate the expectations of $D_{query}(S_i)$ or $D_{event}(S_i)$ as follows: for any event/query imposed inside the replicate storage region S_i, as it only needs to find a nearest storage node inside S_i, the expected distance is less than half the cell size, which is $d/2$; for any event/query imposed outside the replicate storage region S_i, as Fig.3 shows, we calculate the distance according to its quadrant relative to S_i, by summarizing the average distance from arbitrary points to the storage region, thus we can estimate the expectations of $D_{query}(S_i)$ or $D_{event}(S_i)$.

As mentioned before, if we construct a k-d tree based index structure by starting from the root, for each index node at the splitting step, we have two optional strategies to choose: to apply the replicate storage scheme over S_i or conduct further splitting? We know that the energy cost for replicate storage scheme over S_i is:

$$\begin{aligned} E_{rep}(S_i) = {} & f_e \cdot Prob_E(S_i) \cdot E_{event}(S_i) \\ & + f_q \cdot Prob_Q(S_i) \cdot E_{query}(S_i) \end{aligned} \tag{11}$$

The energy cost $E_{rep}(S_i)$ includes the overall event insertion cost, as we know the event access frequency for S_i is $f_e \cdot Prob_E(S_i)$ and the energy cost for each insertion is $E_{event}(S_i)$, and it also includes the overall query diffusion cost with query access frequency $f_q \cdot Prob_Q(S_i)$ and energy cost for each query forwarding $E_{query}(S_i)$.

And consider the k-d tree based index structure, let us denote $T_{lchild}(S_i)$ as the zone associated with left child of S_i, and $T_{rchild}(S_i)$ as the zone associated with its right child, and we utilize $E(S_i)$ to represent the optimized energy cost for region S_i, thus the energy cost for further splitting can be represented as:

$$E_{split}(S_i) = E(T_{lchild}(S_i)) + E(T_{rchild}(S_i)) \tag{12}$$

Therefore the optimized energy cost for region S_i is of the following form:

$$E(S_i) = \min\{E_{rep}(S_i), E(T_{lchild}(S_i)) + E(T_{rchild}(S_i))\} \tag{13}$$

Based on the above analysis we note that the optimized k-d tree structure satisfies the following property: for an optimum k-d tree structure, its sub-tree structures are also optimum. Therefore this optimization problem can be solved by dynamic programming that works from the bottom to the top along the tree.

C. Constructing Optimized Index Structure based on Dynamic Programming

We construct the k-d tree from bottom to top, which is in post-order. Assume the maximum height of the k-d tree is h, initially consider a full k-d tree with all leaf nodes at the hth level, each leaf node correspond to a unit cell, thus the number of nodes over the k-d tree is $n = 2^{h+1} - 1$. For each two nodes i and j within one sub-tree, we denote S_i and S_j as their respective regions, and we denote $S_{i,j}$ as the super-region composed of the two regions, which is corresponding to their parent node. We will try to merge two split nodes along the tree if the energy cost for replicate storage scheme is lower than the sum of each split nodes.

Algorithm 1 depicts the pseudo code of this dynamic programming based algorithm. Assume that the n nodes in the k-d tree are labelled using the post-order. A table $E[1...n]$ is used to hold the minimum energy cost of each node which corresponds to a specific storage region S_i, here for ease of presentation we denote it as $E(S_i)$. In the algorithm, line 3 computes the energy cost $E(S_i)$ for all leave nodes and lines 6-10 compute the minimum energy cost $E(S_{i,j})$ for the remaining internal nodes in post-order. As there're $O(n)$ entries of energy cost to compute according to all the nodes over the initial full k-d tree, and at each step one exact entry is computed, the time complexity of Algorithm 1 is $O(n)$, where n is the number of nodes over the initial full k-d tree.

For implementation of optimized index structure construction, first the sink calculates the event data distribution and range query distribution according to a training set by leveraging the methodology proposed in Section III, and further calculates optimized parameters for the optimized k-d tree based on dynamic programming, then it broadcasts the index structure information to the sensor network, after receiving this information, those selected storage nodes of each cell will confirm their storage roles which includes corresponding storage schemes and the multi-dimensional ranges of data they will store, and each sensor node will know where to store/find the relevant ranges of data according to the locally stored index structure.

Algorithm 1 Constructing k-d Tree based Index Structure with Optimized Granularity

1: Initially construct a k-d tree as a full binary tree with maximum height of h, each leaf node denotes a unit cell.
2: **for** each leaf node associated with region S_i **do**
3: $E(S_i) = f_e \cdot Prob_E(S_i) \cdot E_{event}(S_i) + f_q \cdot Prob_Q(S_i) \cdot E_{query}(S_i)$
4: **end for**
5: **for** all remaining nodes $S_{i,j}$ with corresponding children nodes S_i and S_j, in post-order **do**
6: $\{cost1$ is associated with the replicate storage scheme for region $S_{i,j}\}$
7: $cost1 = f_e \cdot Prob_E(S_{i,j}) \cdot E_{event}(S_{i,j}) + f_q \cdot Prob_Q(S_{i,j}) \cdot E_{query}(S_{i,j})$
8: $\{cost2$ is associated with the split storage scheme for region $S_{i,j}\}$
9: $cost2 = E(S_i) + E(S_j)$
10: $E(S_{i,j}) = \min(cost1, cost2)$
11: **if** $cost1 < cost2$ **then**
12: Apply the replicate storage scheme to region $S_{i,j}$, merge two split nodes S_i and S_j into one node $S_{i,j}$ for the k-d tree.
13: **end if**
14: **end for**

V. Event Storage and Query Processing Mechanisms

A. Inserting Events to Cells

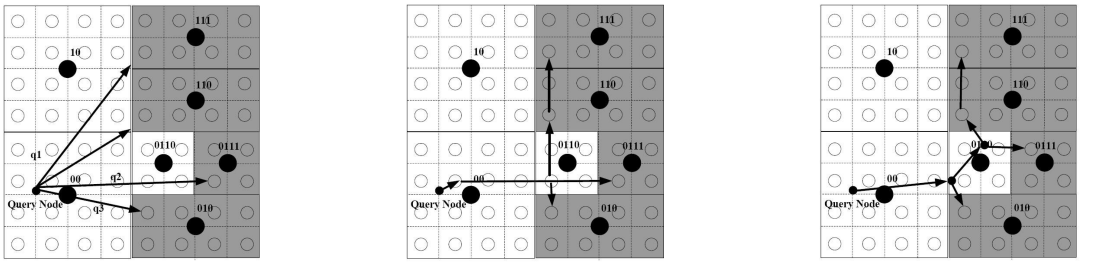
As an event data is a point value over the multi-dimensional data space, it corresponds to a unique zone over the deployment area. When a sensor node generates an event data, it will check the locally stored index structure to find the corresponding zone, then it leverages the GPSR[1] routing scheme to forward the event message to that specified region. As described in Section IV, when the event message reaches the zone, we utilize the Double Ruling scheme [9][10] to distribute the event information to all relevant replicate storage nodes.

B. Resolving and Routing Range Queries

As a range query corresponds to a region over the multi-dimensional data space, thus a conventional range query may cover several zones over the k-d tree based index structure. For example, consider a range query $\langle[0.3, 0.7], [0.5, 0.6], [0.4, 0.8], [0.7, 0.9]\rangle$ over the example k-d tree depicted in Fig.2(a), the storage regions of zone code 110,111, 010, 0111 will be involved. Therefore how to route these sub-range queries to those specified storage nodes in an energy efficient approach is the problem we are trying to address in this section. As the corresponding query replies can be sent back to query node along the query routing paths, thus here we only consider the query diffusion cost, for the overall cost is actually twice the query diffusion cost.

Basically there are three strategies to route these sub-queries. The first strategy is to route sub-queries separately to corresponding zones by selecting nearest storage nodes for the query node. As Fig.4(a) shows, for each relevant storage region S_i associated with sub-query q_i, the query node selects a nearest storage node from those replicate storages, and respectively sends a sub-query message to the corresponding storage node. In this scheme, no routing paths are shared.

The second strategy leverages *Double Ruling* based routing approach. As Fig.4(b) shows, the query node first imposes a routing path horizontally, and at specific positions of the



(a) Separate routing strategy (b) Double ruling based routing strategy (c) Steiner tree based routing strategy

Fig. 4. Routing strategies for range query forwarding

horizontal path it further imposes routing paths vertically to reach those corresponding storage regions. This routing scheme uses the horizontal routing path as the backbone path, thus it brings opportunities to sufficiently share the horizontal routing path, however, as it only utilizes the rectangular paths, which may not be efficient enough when compared with the shortest path.

Hence according to the analysis over the pros. and cons. of the above two strategies, as our ultimate goal is to find the connecting paths with minimum routing cost, given the query nodes and specified storage nodes, we can reduce the above problem into the *Euclidean Minimum Steiner Tree*[11] problem. Based on the theory of Euclidean Steiner tree, we set the query node and those specified storage nodes as terminal points, we then construct a Steiner tree according to these terminal points by selecting specific sensor nodes as the Steiner points, thus to sufficiently share the routing path and achieve the minimum routing cost for best energy efficiency. Fig. 4(c) shows the Steiner tree based routing paths.

C. Optimized Routing Scheme for Query Forwarding

We have reduced the optimized query forwarding problem into the *Euclidean minimum Steiner tree* problem, actually as the problem is NP hard, so polynomial-time heuristics are desired. There're a lot of research works which presents heuristics algorithms to approximate the optimized Steiner tree[11][12][13] , and most of them are mainly solved by first constructing a *Minimum Spanning Tree*, for example, Dreyer et.al. proposed two heuristics for the Euclidean Steiner tree problem[12], which includes the *Steiner Insertion Algorithm* and the *Incremental Optimization Algorithm*. The *Steiner Insertion Algorithm* systematically inserts Steiner points between edges of the minimal spanning tree meeting at angles less than 120 degrees, performing a local optimization at the end. The *Incremental Optimization Algorithm* begins by finding the Steiner tree for three of the fixed points. Then, at each iteration, it introduces a new fixed point to the tree, connecting it to each possible edge by inserting a Steiner point, and minimizes over all connections, performing a local optimization for each. In this paper, we leverage *Steiner Insertion Algorithm* as approximation method due to its simplicity and low computing complexity. Readers can refer to [12] for detail algorithms.

Assume for each specified zone S_i, it has a set of replicate storage node $\{s_{i,1}, s_{i,2}, ..., s_{i,m}\}$, here m is the number of redundancy of S_i, which we can also denote as $|S_i|$, we propose Algorithm 3 to compute the optimized routing path for query diffusion by leveraging the approximation algorithms for Euclidean Minimum Steiner tree problem.

Algorithm 2 Computing the optimized routing path for query diffusion

Input: Query node s_q, storage zone $S_1, S_2, ..., S_n$ specified by range query Q
Procedure:
Let $opt = \infty$, $steinset = \phi$
for each storage zone $S_i (1 \leq i \leq n)$ **do**
  Enumerate each storage node $s_{i,j}$ from S_i, $(1 \leq j \leq |S_i|)$
  $s_i = s_{i,j}$
  $[cost, S]$=*ComputeSteinerTree*$(s_q, s_1, ..., s_i, ..., s_n)$
  if $(cost < opt)$ **then**
    $opt = cost$
    $steinset = S$
  end if
end for
Output: opt, $steinset$

As each storage zone covers several storage nodes as the candidate terminal points for the optimized solution, the above algorithm enumerates all permutations of storage nodes $(s_1, s_2, ..., s_n)$ from those specified zones $S_1, S_2, ..., S_n$, and leverage the function *ComputeSteinerTree* to approximate the optimized Steiner tree according to each permutation, among these candidate Steiner tree solutions we finally choose the one which gives the minimum cost as the optimized Steiner tree solution. As here we leverage the *Steiner Insertion Algorithm* as the approximation method, which has the time complexity of $O(n^3)$, consider the number of permutations $M = \prod_{1 \leq i \leq n} |S_i|$, we have the computing complexity as $O(M \cdot n^3)$. And the routing cost for query diffusion is equal to the overall lengths of the optimized Steiner tree.

Consider the distributed implementation of this optimized routing scheme, as each sensor node has the global information of the index tree, for the query node, it first calculates the optimized Steiner tree according to the specified range query, then it route the messages of sub-queries along the Steiner tree

by leveraging the GPSR routing scheme, thus to reach those corresponding storage regions, then the query results will be sent back along the original routing paths.

VI. Simulation Results

In this section, we use simulation to compare the performance of our optimized storage scheme against conventional storage schemes for multi-dimensional range queries. We conduct the simulation based on the grid based system, we divide the sensor network into $n \times n$ cells (we set 8 as the default value for n), and each cell represents a unit storage area, we normalize the cell size to 1. For performance comparison we separately build DIM's zone tree, the fully replicate storage scheme, and our optimized k-d tree over the network. Here we assign a unique zone for each of the cell units in DIM's zone tree, which infers to be a fully split approach over the cell based network, and we construct the fully replicate storage scheme as another extreme situation, which stores all sensor data into each of the cell units, thus all the cells are replicate storage units inside the unique zone.

And we utilize a training set of events and range queries to derive the event and query distributions, thus to construct our optimized k-d tree based index structure, as Fig.5 depicts, we use the Gaussian-like distribution to generate 5-dimensional data and queries for the training set and we normalize all the attribute values into the range [0,1], for range queries, we set the query range for each dimension within range [0,1]. And we also generate a similar set of events and range queries to evaluate the performance of various multi-dimensional storage schemes, we set the default number of event frequency f_e and query frequency f_q to 1000.

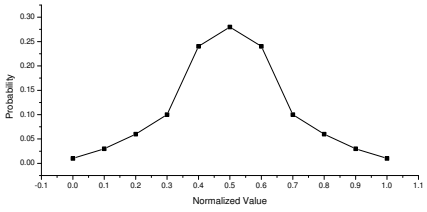


Fig. 5. The Gaussian-like distribution of multi-dimensional data

A. Comparing performance for various multi-dimensional storage schemes

In this section, we evaluate the energy efficiency for various multi-dimensional storage schemes. Fig.6 depicts the optimized k-d tree and corresponding storage zones over the cell based network, this optimized index structure is constructed according to the event and query training set generated in terms of the Gaussian-like distributions.

We now compare the average event insertion cost and average query delivery cost. Fig.7(a) shows the average event insertion cost for DIM, the fully replicate storage scheme, and optimized K-d tree. As the network size increases, the fully replicate storage scheme always achieves the most energy consumption as it requires the event message to update all $n \times n$ replicate storage cells. And DIM always achieves the least energy consumption as it requires the event message to update one unique corresponding storage cell. For our optimized k-d tree storage scheme, as the zone specified by the event may include one or more replicate storage cells, thus its energy consumption is between the above two extreme situations, and unlike the fully replicate storage scheme, it grows slowly as the network size increases.

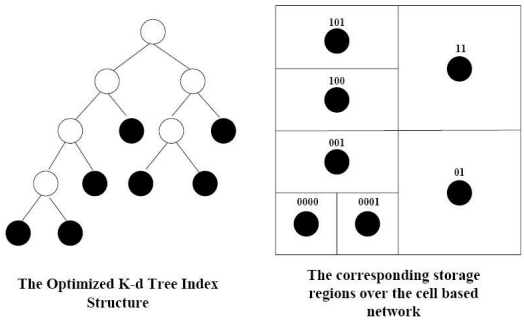


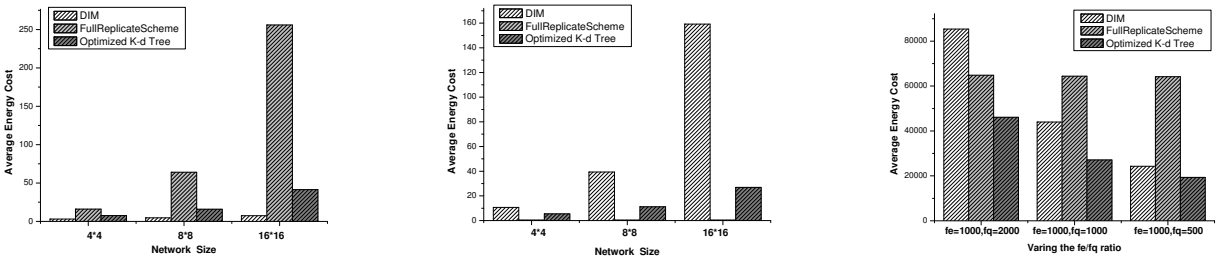
Fig. 6. The optimized k-d tree and corresponding storage zones over the cell based network.

Fig.7(b) shows the average query diffusion cost for the above three storage schemes. Note that DIM always achieves the most energy consumption and the fully replicate storage scheme always achieves the least energy consumption, because for one specific range query, DIM may split the query into sub-queries and map them to quite many storage zones while the fully replicate storage scheme simply maps the query to one unique zone at the top level with replicate storages, thus its energy cost is nearly negligible. And similar to the former case, our optimized k-d tree storage scheme gains a tradeoff between the two extremes, and unlike DIM, it grows slowly as the network size increases.

As Fig.7(c) shows, we further compare the overall energy cost for the above three schemes. For the situation $f_e = 1000, f_q = 2000$, DIM achieves the most energy cost as DIM's query diffusion cost is the largest among the three schemes, although its event insertion cost is rather small. As the f_e/f_q ratio increases, the fully replicate storage scheme achieves the most energy cost due to its large event insertion cost. For all the above three situations, our optimized k-d tree storage scheme always achieves the least overall energy cost, as it gains an optimized tradeoff to sufficiently control both the event insertion cost and the query delivery cost. For the situation $f_e = 1000, f_q = 1000$, our optimized k-d tree storage scheme reduces 39% energy cost than DIM and further reduces 58% energy cost than the fully replicate storage scheme.

B. Comparing performance for various query diffusion approaches

In this section we compare the energy cost for various query diffusion approaches: the normal separate routing approach,



(a) Average event insertion cost comparison. (b) Average query diffusion cost comparison. (c) Overall energy cost comparison.

Fig. 7. Comparing performance for various multi-dimensional storage schemes: DIM, Full Replicate Storage Scheme, and Optimized K-d Tree

the double ruling based routing approach and the Steiner tree based routing approach. As the query replies are sent back along the query paths, here we only consider the query diffusion cost, the overall query processing cost is thus double the cost of query diffusion. We leverage Prim's algorithm[14] to calculate the minimum spanning tree (MST) so as to approximate the Steiner tree. Fig.8 depicts the query diffusion cost for various schemes. Note that for all three storage schemes the straight separate routing approach achieves the most energy cost and the Steiner tree based routing approach achieves the best energy efficiency. For DIM the Steiner tree based routing approach reduces 61% energy cost than the straight separate routing approach and reduces 26% energy cost than the double ruling based routing approach, for the optimized k-d tree storage scheme the numbers are respectively 50% and 10%, for the fully replicate storage scheme, as the query message only has to be forwarded to the nearest storage node, the energy costs for the three approaches are equally negligible.

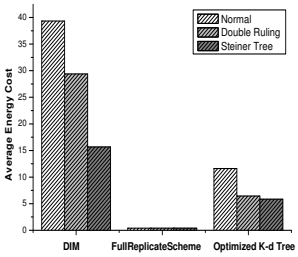


Fig. 8. Query diffusion cost for various schemes.

VII. Conclusion

This paper presents the design of a decentralized storage scheme for multi-dimensional range queries over sensor networks. We build a distributed index structure, which is based on k-d tree, so as to efficiently map high dimensional event data to a two-dimensional space of sensors while still preserving the proximity of events. We propose a dynamic programming based methodology to control the granularity of the index structure in an optimized approach, and an optimized routing scheme for range query processing to achieve best energy efficiency.

VIII. Acknowledgement

This work is supported by the China NSF grant (60573132, 60573106, 60873026), the China 973 project (2009CB320705,2006CB303000). We would like to express our thanks to Yingpei Zeng and Jue Hong, for their helpful comments and for assistance with literature.

References

[1] R. Sylvia, K. Brad, S. Scott, E. Deborah, G. Ramesh, Y. Li, and Y. Fang, "Data-centric storage in sensornets with ght, a geographic hash table," *Mobile Net-works and Applications*, vol. 8, no. 4, pp. 427–442, 2003.

[2] S. Scott, R. Sylvia, K. Brad, G. Ramesh, and E. Deborah, "Data-centric storage in sensornets," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 137–142, 2003.

[3] G. Deepak, E. Deborah, and H. John, "Dimensions: Why do we need a new data handling architecture for sensor networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 143–148, 2003.

[4] B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin, "Difs: a distributed index for features in sensor networks," *Ad Hoc Networks*, vol. 1, no. 2-3, pp. 333–349, 2003.

[5] J. L. Bentley., "Multidimensional binary search trees used for associative searching," *Communicaions of the ACM*, vol. 18, no. 9, p. 475C484, 1975.

[6] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *Sensys*, 2003.

[7] M. Aly, K. Pruhs, and P. K. Chrysanthis, "Kddcs: a load-balanced in-network data-centric storage scheme for sensor networks," in *CIKM*, 2006.

[8] Y. C. Chung, I. F. Su, and C. Lee, "Supporting multi-dimensional range query for sensor networks," in *ICDCS*, 2007.

[9] X. Liu, Q. Huang, and Y. Zhang, "Combs, needles,haystacks: balancing push and pull for discovery in large-scale sensor networks," in *Sensys*, 2004.

[10] R. Sarkar, X. J. Zhu, and J. Gao, "Double rulings for information brokerage in sensor networks," in *MOBICOM*, 2006.

[11] F. Hwang, D. Richards, and P. Winter, "The steiner tree problem," *GI Jahrestagung*, vol. 1, no. 2, pp. 370–374, 2003.

[12] D. R. Dreyer and M. L. Overton, "Two heuristics for the euclidean steiner tree problem," *Annals of Discrete Mathematics*, vol. 53, 1992.

[13] M. Diane and J. Plesnik, "Three new heuristics for the steiner problem in graphs," *Acta Math. Univ. Comenianae*, vol. 60, pp. 105–121, 1991.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*.