Fractionally Cascaded Information in a Sensor Network

Jie Gao[*] Leonidas J. Guibas[*] John Hershberger[†] Li Zhang[‡]

ABSTRACT

We address the problem of distributed information aggregation and storage in a sensor network, where queries can be injected anywhere in the network. The principle we propose is that a sensor should know a "fraction" of the information from distant parts of the network, in an exponentially decaying fashion by distance. We show how a sampled scalar field can be stored in this distributed fashion, with only a modest amount of additional storage and network traffic. Our storage scheme makes neighboring sensors have highly correlated world views; this allows smooth information gradients and enables local search algorithms to work well. We study in particular how this principle of fractionally cascaded information can be exploited to answer range queries about the sampled field efficiently. Using local decisions only we are able to route the query to exactly the portions of the field where the sought information is stored. We provide a rigorous theoretical analysis showing that our scheme is close to optimal.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: information storage and retrieval—*information search and retrieval*; F.2.2 [**Theory of Computation**]: analysis of algorithms and problem complexity—*non-numerical algorithms and problems*

General Terms

Algorithms, Design

[*]Department of Computer Science, Stanford University, Stanford, CA 94305, USA Email: {jgao, guibas}@cs.stanford.edu
[†]Mentor Graphics, 8005 S.W. Boeckman Road, Wilsonville, OR 97070, USA Email:john_hershberger@mentor.com
[‡]Hewlett-Packard Labs, 1501 Page Mill Road, Palo Alto, CA 94304, USA Email: l.zhang@hp.com

Keywords

Sensor networks, information aggregation and storage, fractional cascading, range searching

1. INTRODUCTION

Sensor network technology is in a rapid state of evolution, especially at the hardware level. Sensors are becoming smaller, more power efficient, and less expensive [5]. Networking for sensor networks is also quickly developing, with certain protocols such as directed diffusion [11] already widely deployed and sensor-net specific adaptations of the network stack drilling down all the way to the MAC layer [23, 10]. But for sensor nets to attain their true potential, equal strides must be made at the application level, facilitating the deployment and use of a sensor network for a variety of purposes by many simultaneous and geographically distributed users. To this end, a more abstract view of the sensor network is useful, and the view that has received the most attention to date is that of the sensor network as *a distributed database* [8].

The advantage of the database approach is that it offers a separation between the logical view (naming, access, operations) of the data held by the sensor net and the actual implementation of these operations on the physical network. Though any such abstraction comes with some loss of efficiency, it compensates by increased ease of use. Sensor net users can focus on the logical structure of the queries they want to ask and are relatively isolated from the details of physical storage and data networking on the volatile physical infrastructure of the network (sensors can fail, links come and go, etc.).

Standard distributed database technology, however, cannot be ported as-is to sensor networks because both the logical structure of the data and the economics of the physical medium are different. At the logical level, one major difference is that sensor data are typically measurements from the physical world and as such there is always the potential of error. Thus exact queries are not so meaningful in the sensor network context—instead it is more appropriate to use *probabilistic queries* [6], or *range queries*, where we ask that the values of certain attributes lie in certain ranges. Furthermore, sensors often sample a physical quantity at discrete time intervals and thus generate a stream of data. Whatever index structure we plan to build must therefore be able to accommodate a stream of continuous updates, as we go forward in time. Furthermore, the cost of processing a query is likely to be dominated more by the communication costs of getting to all the relevant data than by the data processing or aggregation cost. While in a classical database we worry about what and how much to store in an index, in the sensor net setting the dominant worry is where to store the distributed index.

In this paper we investigate a new lightweight distributed mechanism for storing data in a sensor network that allows range queries injected anywhere in the network to be served efficiently. The key idea of our approach is to store at each sensor information about data available elsewhere in the network, but in such a way that a sensor knows only a "fraction" of the information from distant parts of the network, in an exponentially decaying fashion by distance. The precise way in which information is to be subsampled, compressed, or aggregated to meet this constraint will be application dependent. This accomplishes three goals simultaneously:

- the total amount of information duplication across all sensors is kept small, because of the geometric decrease with distance,
- the communication cost required to build this index and its update cost remain reasonable, as on the average information travels only short distances, and
- neighboring sensors have highly-correlated world views; this allows for smooth information gradients and enables local search algorithms to work well.

Because our method is effectively a broadcast with geometric decay of the information, we call the technique *fractional cascading.*

Although several prior works, as discussed below, have also given distributed indices for range queries in sensor networks, our approach highlights an important aspect of queries in a sensor network that has not been discussed much up to now. This is the issue of *locality*: when we make queries to a sensor network, we are more likely to ask about recent events in nearby places. Fractional cascading automatically adapts the resolution at which information is stored so that more detailed information is available about data obtained in the spatio-temporal locality of the sensor where the query is injected—but without sacrificing the ability to query distant regions or times as well. Furthermore, this is accomplished with load-balancing across the entire network field.

To illustrate the potential of fractional cascading we discuss in this paper the simple scenario of a dense uniform sensor network sampling a smooth scalar field, such as temperature. A typical range query for us would be to give a query region (specified as a rectangle parallel with the axes) plus a threshold temperature and then ask for all the "hot spots", i.e., the sensors in the query region where the temperature is above this threshold. We perform a theoretical analysis of such queries and give asymptotic upper and lower bounds on the cost of serving these queries. If we have n sensors then the overall storage taken by the fractionally cascaded index is $O(n \log n)$. If there are k hot sensors in a query region of area A and perimeter P at distance D from the sensor where the query is injected, then the cost of the query can be expressed as some fixed overhead cost $f(D, P, A, n)$ plus an additional cost $f(D, P, A, n, k)$ that depends on the number of hot spots. The parameters, D, P, A, represent the locality, i.e., how far away the information about the query is from the sensor where the query is injected. Roughly speaking, our upper bounds show that fractional cascading performs with overhead $O(D + P)$ up to logarithmic factors, and the cost related to the number of hot spots is $O(\sqrt{Ak})$. In a simplified model we can argue that these bounds are essentially best possible for any method that uses comparable storage.

Since the temperature field is smooth, hot spots in general will tend to be clustered. A variant of our range query then might ask for the isocontours of the temperature field at the desired threshold. If s is the number of sensors bordering these contours, we show that we can exploit this coherence in the field and return these boundary sensors at a cost of $O(\sqrt{As})$, without having to visit all the hot sensors inside the contours.

The idea of our method is to cover the underlying space by a quad-tree and then store the summarized information of each quadrant with the sensors in the sibling quadrants. While our method resembles previous methods, we are the first to rigorously analyze the performance of such a method as well as analyze the limitations of any distributed range query method.

2. PRIOR WORK

It seems that any kind of index on distributed data requires some type of hierarchical structure that aggregates information from different portions of the attribute space or from different regions of the network. Prior work in range searching for sensor networks has addressed a number of important issues in constructing such hierarchies. One of these is efficient data summarization. Wavelet transforms provide one way to compress and summarize information for both temporal and spatial signals, and are widely used in signal and image processing. Wavelet encodings can be used with a standard hierarchical structure, such as a spatial quad tree, to provide data summarization across spatial regions of the network. More detailed information can be accessed as needed by a top down traversal of the hierarchy tree, using geographic routing [12] to visit the sensors holding the relevant information. This has been used in a number of prior systems, including DIMENSIONS [7] and TinyDB [16, 17, 18].

Any hierarchical structure has the problem that sensors that hold information corresponding to high levels in the hierarchy become traffic bottlenecks, as they need to be accessed by many queries. This overload can be avoided by structured replication of information [21]. In the case of a scalar field such as our temperature, another solution is to partition the information about large geographic regions into subsets according to smaller allowed ranges of the field value, and store these subsets in different nodes. This is the approach taken in the DIFS system [9], where each node in a quad-tree has multiple parents, according to a finer partition based on smaller field ranges. Thus the wider the spatial extent an index node knows about, the more constrained the value range it covers.

Many of the above systems use hashing to select geographic locations for the sensors that will hold parts of the index, according to the attribute values stored in that part of the index. This is the original idea of geographic hash tables [21]—a hash function based on data type and value is used to select where the data will be stored, so as to provide a rendezvous point for those producing and those seeking the data. In the DIM system [15] a locality preserving hash function is used to map portions of a multidimensional attribute space to sensors so that all data needed to answer a range searching query can be located conveniently.

Although in DIMS there is an attempt to place the index for nearby portions of the attribute space in nearby sensors, none of these systems explicitly tries to place the index for information close to where the information was generated. This is one of the novel aspects of our fractionally cascaded index that should benefit queries that exhibit locality.

This principle of fractionally cascaded information was first introduced in a very different context, in our work on speeding up geometric data structures [2, 3]. In that context each node stores an ordered list of keys, shares a well-distributed sample of that list with its neighbors, a sample of the sample with the neighbors two away, and so on. Each node then simply merges together all lists it owns or receives. As shown in [2], without significantly increasing storage, this method can be used to speed up a key search in a sequence of nodes along a path by making the key lists of nearby nodes highly correlated. The principle of fractional cascading has been widely used by the computational geometry community since then [4, 19, 22].

Similar ideas have arisen in other contexts as well. *Fisheye state routing* [20] is a proactive routing protocol that reduces the frequency of topology updates to distant parts of the network. Also, the *spatial gossip protocol* [13] is a gossip protocol in which each node in a peer-to-peer network chooses to talk to another node with a probability that decreases polynomially with the distance between the two nodes.

3. RANGE SEARCH QUERIES

In this section, we provide a formal description of the problem we are to solve. Assume there is a continuous scalar field, say temperature, and a set of n sensors S in the plane; each sensor $s \in S$ reads the temperature information $t(s)$ at its location. We assume the sensors are uniformly and densely distributed in a square. More formally, suppose that the communication range of each sensor is normalized to 1. Two sensors can directly communicate with each other if they are within communication range. Then, we require that each point in the square must be within distance 1 to some sensor and the minimum number of hops of the path to connect any two sensors p, q is at most a small constant times the Euclidean distance between p, q. The communication between two sensors is done by multi-hop routing, for example, a geographic forwarding scheme. We assume that the routing scheme gives reasonably short paths, e.g., comparable to the Euclidean distance.

We consider the following range query problem in such a sensor network: a query consists of three components (q, R, T) where q is the sensor that initiates the query, R is an $a \times b$ rectangular range within the square, and $T = [T_1, T_2]$ is a temperature range. As in standard geometric range query problems [1], we study three versions of the problem:

- *counting problem*: we want to know the number of sensors with temperature reading between T_1 and T_2 in the region R;
- *reporting problem*: we want to collect a list of sensors inside the range R with temperature readings inside the temperature interval $T = [T_1, T_2]$. In some applications, we may need to construct a spanning tree to connect the query sensor and the qualified sensors;
- *isometric contour problem*: assuming the approximate temperature field can be constructed by interpolation from the sample readings of the sensors, we want to get the isometric contour with temperature reading T inside the query range R.

The answer of each query should be returned to the query sensor. The cost of a query is measured by the total number of steps, including both the communication and computation it requires to get the answer.

The most obvious solution to answer such a query is by sending the query to all the sensors in the network and having a sensor respond if its position is within the query range and its sensor value is within the query interval. The answers can be combined as they propagate back to the query sensor. A little bit more sophisticated scheme is to flood the query only to the query range with some geographic routing algorithm. The advantage of such a scheme is its simplicity: there is no need for any precomputation, and each sensor stores only its own sensor value. However, the query cost of such an algorithm can be high: it is proportional to the number of sensors in the query range. In the case when one needs to perform multiple queries, it is often more cost effective to first perform an information aggregation stage to reduce the query cost.

The most straightforward information aggregation scheme is to construct *global warehouses*, i.e., let all the sensors send their data to some "warehouse" nodes. We then build a standard range query data structure at each warehouse. Each query is sent to a warehouse and answered by using the precomputed data structure. In this approach, the query can be answered very efficiently, given that the query sensor is not too far from some warehouse. However, such an approach requires warehouses capable of holding all the data and the construction of the range query data structure. Further, we need to distribute multiple warehouses evenly in the network, so that each query sensor is guaranteed to be not too far from a warehouse.

In this paper, we provide a solution in which every sensor stores only a small amount of aggregated information, and the query can be answered much more efficiently than without aggregation. Furthermore, we show that our method is quite close to the optimum. A formal comparison between our method and the flooding and global warehouse methods can be found at the end of this paper.

4. LOWER BOUNDS

Before we describe the fractional cascading scheme, we first show several query time lower bounds that hold when each sensor is allowed to store only a small amount of information. Our lower bounds are based on information-theoretical arguments. By those lower bounds, we can understand the limitation of any distributed range query algorithm when each sensor is only allowed to have limited information storage.

For the lower bounds, it suffices to consider a simpler problem in which the sensors are located on a regular $\sqrt{n} \times \sqrt{n}$ grid, and the sensor value is binary, 0 or 1. A sensor is "hot" if its value is 1. A query is to count the number of hot sensors in a query range. We assume that each sensor can store at most m bits after the information aggregation stage. In practice, m needs to be small. We can think it is bounded by $O(\log n)$ or $O(\operatorname{polylog} n)$.

There are two factors that affect the query complexity. One is the distance from the query sensor to the query range, and the other is the size of the query range. We consider those two factors separately by considering two types of queries. In the Type I queries, each query rectangle singles out a sensor, i.e., the query is a pair of sensors (q, r), and we wish to get the value of r to q. In the Type II queries, we require the query range R to contain the query sensor q.

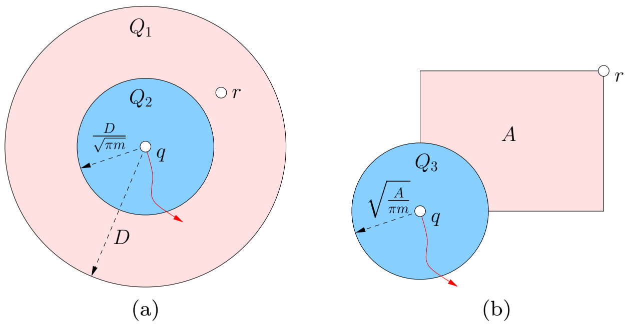

Figure 1. (a) Type I query; (b) Type II query.

For the Type I query, the naïve method of sending a message from r to q takes $O(D)$ steps, where D is the shortest distance between q and r. We will show that this is close to the best we can hope for. Let Q_1 denote the set of sensors that are within distance D from q and Q_2 the set of sensors within distance $\frac{D}{\sqrt{\pi m}}$ from q. Then we know that the number of sensors in Q_1 is more than D^2. On the other hand, there are at most D^2/m sensors in the region Q_2 and therefore at most D^2 bits stored in the sensors in Q_2. Thus, there exist two different assignments t_1, t_2 of sensor values to the more than D^2 sensors in Q_1 that result in the same information being stored in the sensors in Q_2. Let r be a sensor at which $t_1(r) \neq t_2(r)$. Then the query (q, r) has to take $\Omega(\frac{D}{\sqrt{m}})$ steps, because otherwise we would not be able to distinguish $t_1(r)$ and $t_2(r)$ by querying only the sensors in Q_2.

Now we consider Type II queries. For any two sensors q, r, denote by $R(q, r)$ the rectangle with q, r as two diagonally opposite vertices. Suppose that t_1, t_2 are any two different assignments of the sensor values to the sensors in R. It can be verified that there always exists a sensor $r \in R$ such that $t_1(r) \neq t_2(r)$ but for all the other sensors $r' \in R(q, r)$, $t_1(r') = t_2(r')$. Thus, the query $(q, R(q, r))$ will receive different answers for the assignments t_1 and t_2. Therefore, if the query range has area A, i.e., contains A sensors, then there are 2^A different assignments that need to be distinguished. By the same argument as before, we need an amount of memory at least equal to A to differentiate those different assignments. Let Q_3 be the set of sensors within distance $\sqrt{\frac{A}{\pi m}}$ from q. So q has to ask a sensor outside Q_3 to answer the query. This requires $\Omega(\sqrt{\frac{A}{\pi m}})$ steps in the worst case.

By combining the above two cases, we have that the lower bound is $\Omega(\max(\frac{D}{\sqrt{m}}, \sqrt{\frac{A}{m}})) = \Omega(\frac{D+\sqrt{A}}{\sqrt{m}})$ where D is the shortest distance from the query node to the query range, and A the area of the query range.

Last, we note that if we require that all the hot sensors in the query range be visited at least once, then there is an intrinsic lower bound of $\Omega(\sqrt{kA} + P)$ for just touring k hot sensors, where P and A are, respectively, the perimeter and the area of the query range. The lower bound is realized by distributing k hot sensors evenly in the square. To summarize, we have that:

Theorem 4.1. *Suppose that each sensor stores at most m bits. Let D denote the shortest distance from the query sensor to the query range and P, A denote the perimeter and the area of the query range, respectively. Then in the worst case, it takes $\Omega(\frac{D+\sqrt{A}}{\sqrt{m}})$ steps to answer the query. If we need to connect the query sensor to the k hot sensors, then it requires $\Omega(D + P + \sqrt{kA})$ steps.*

We should note that the lower bounds above are fairly general, since they depend only on the amount of information stored at each sensor and are independent of how the information aggregation is done. If we take into account the communication and computation cost in the aggregation stage, we may obtain even higher lower bounds. However, these lower bounds are actually quite tight even we take the communication and computation costs into account — in the next section, we show that for the reporting problem and for the counting problem in ranges with constant bounded aspect ratio, the query cost of our algorithm matches the lower bounds up to a multiplicative logarithmic factor.

5. FRACTIONALLY CASCADED INFORMATION

We describe our method for reporting query $(q, R, [T, \infty))$, i.e., report all the sensors inside range R with temperature higher than T. The other types of queries will be discussed later as variants.

5.1 Aggregation algorithm

We first build a standard (virtual) quadtree T on the sensors. The root node is associated with a bounding box that covers all the sensors. The quadtree is defined recursively such that each internal node is divided into four equal size squares. Every leaf node contains at most one sensor. Denote by $p(u)$ the parent node of a quadtree node u. We also associate with each node u in the quadtree its bounding box $B(u)$. Without loss of generality, we assume the smallest square of the quadtree is a unit square. The side length of $B(u)$ is denoted as $L(u)$. The side length of the root node is denoted by L. A quadtree node u of height i has a bounding box $B(u)$ with side length 2^i. Since the number of sensors is n and the sensors are uniformly distributed, the height of the quadtree is $O(\log L) = O(\log n)$.

For a centralized algorithm, the quadtree can be used to answer range search queries efficiently. Each internal node u of the quadtree saves the maximum temperature, denoted by $t(u)$, among all the sensors in its subtree. Given a query rectangle R, we traverse the quadtree starting from the root. If the current node's maximum temperature is lower than the threshold or the current node's bounding box is completely outside the query range, we discard it. If the current node is a leaf node, we report the sensor. Otherwise we visit its 4 children recursively.

The centralized approach does not fit the sensor network scenario in several ways. Firstly we don't want to have any sensor storing the whole quadtree, for reasons of fault tolerance. Furthermore, this approach is not scalable, since the sensors are normally memory constrained. Second, in a sensor network, getting the value of a certain sensor requires communication. If the query range is near the source where the query is injected, we expect to answer the query locally, rather than communicating with a far away server.

In the following we show how to store the virtual quadtree implicitly on the sensors, in a fractionally cascaded fashion. In the remainder of the paper, when we write "quadtree node" we are referring to the logical quadtree structure. The value of a quadtree node is saved at physical sensors, with the storage scheme as follows. For each quadtree node u, the maximum temperature $t(u)$, is stored in all the sensors in $B(p(u))$, where $p(u)$ is the parent node of u. We calculate $t(u)$ in a bottom-up approach. We only need to show how to calculate the maximum temperature of a quadtree node u, when the maximum temperatures of the 4 children nodes v_i, $i = 1, 2, 3, 4$, of u have been calculated and stored. According to the induction hypothesis, every sensor in $B(u)$ has the maximum temperatures of all the children nodes of u. One of them[1], say p in $B(u)$, does the aggregation and calculates the maximum temperature in $B(u)$ by $t(u) = \max_{i=1}^{4} t(v_i)$ and saves the information in all the sensors in u and u's siblings by broadcasting. In short, each sensor p saves the maximum temperature $t(u)$, where u is a sibling node of a quadtree node on the path from p to the root. A sensor's view of the world is divided into tiles. A sensor saves only the summarized information of each tile. The further away the tile is from the sensor, the larger the tile could be. Figure 2 (b) shows an example. We bound the amount of storage and communication cost by the following two theorems.

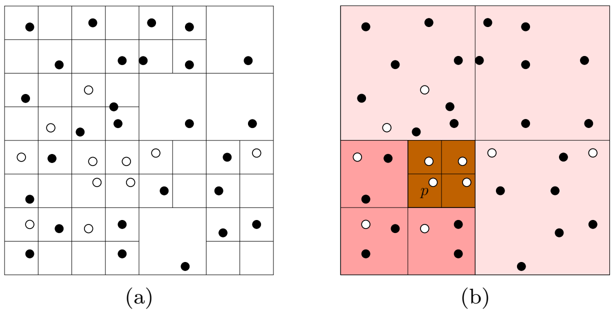

Figure 2. (a) Leaves of the quadtree; (b) a sensor p's view of the world.

Theorem 5.1. *A sensor stores $O(\log n)$ information.*

Proof. A sensor p saves the maximum temperature $t(u)$ for all the quadtree nodes u that are siblings of a quadtree node on the path from p to the root. Since the quadtree has height $O(\log n)$, the total amount of information stored at any sensor is $O(\log n)$. $\square$

Theorem 5.2. *The total communication cost of constructing the implicit quadtree is $O(n \log n)$, where n is the total number of sensors.*

[1]The sensor that does the aggregation can be chosen randomly, or rotated periodically for load balancing.

Proof. The communication cost of calculating the maximum temperature $t(u)$ of a quadtree node u is $O(n(p(u)))$, where $n(v)$ is the number of sensors inside $B(v)$ and $p(u)$ is the parent node of u. The total communication cost is

$$\sum_u O(n(p(u))) = \sum_{i=0}^{\log L} \sum_{L(u)=2^i} O(n(u)).$$

Since $\sum_{L(u)=2^i} O(n(u)) = n$, the total amount of communication is bounded by $O(n \log L) = O(n \log n)$. $\square$

5.2 Query algorithm

For a query range R, we call a node u of the quadtree a canonical piece if $B(u)$ is entirely inside R but $B(p(u))$ is not inside R. The algorithm for answering the query is as follows:

- Send a message from the source to the query range;
- Visit all the canonical pieces;
- For each canonical piece, traverse the subtree to report all the hot sensors.

The following theorem bounds the cost of the reporting query.

Theorem 5.3. *The cost of the reporting query $(q, R, [T, \infty))$ is $O(D + \sqrt{Ak} + P \log P)$, where k is the number of sensors in the answer, D is the length of the shortest path from the source to the query range R, and P and A are, respectively, the area and perimeter of R.*

Proof. Let a, b denote the side lengths of the query rectangle. Suppose that R is represented by the union of a set C of disjoint canonical pieces in the quadtree. For any $u \in C$, we denote by $L(u)$ the side length of $B(u)$ and $k(u)$ the number of hot sensors inside $B(u)$. We bound the cost of visiting all the canonical pieces and the recursive traversal inside each canonical piece separately.

For each canonical piece, we need to visit only one sensor inside its range. We can visit the canonical pieces in a spiral order as shown in Figure 3. Since the largest canonical piece has perimeter no more than $4\min(a, b)$, and there are at most $\log(\min(a, b))$ different sizes of canonical pieces, the cost of visiting all the canonical pieces is

$$\sum_{u \in C} L(u) = \sum_{i=0}^{\log(\min(a,b))} \sum_{u \in C, L(u)=2^i} L(u).$$

One observation is that the canonical pieces with size $L(u) = 1$ must be adjacent to the boundary of the query range. If a canonical piece with size 1 is of distance at least 1 away from the boundary, then it should be able to merge with its neighbors to form a larger canonical piece, thus violating the definition of canonical pieces. Therefore $\sum_{u \in C, L(u)=1} L(u) \leq 2(a+b)$. Recursively we know $\sum_{u \in C, L(u)=2^i} L(u) \leq 2(a+b)$. Thus the cost of visiting all the canonical pieces is $O((a + b)(\log(\min(a, b)) + 1))$.

To bound the communication cost of traversing the canonical pieces, we first prove a lemma.

Lemma 5.4. *The communication cost of traversing the subtree T_u of a canonical piece u is $O(L(u)\sqrt{k(u)})$, if the side length of $B(u)$ is $L(u)$ and the number of hot sensors in $B(u)$ is $k(u)$.*

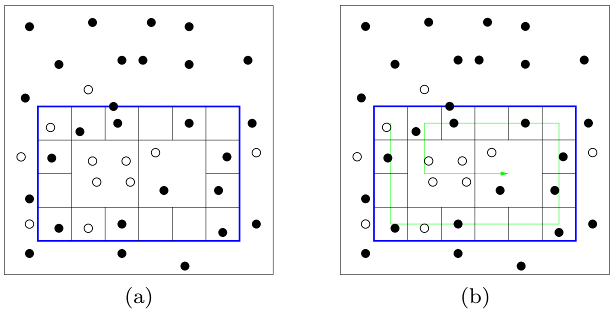

Figure 3. (a) Query range and canonical pieces; (b) Visit all the canonical pieces in a spiral order.

PROOF. We traverse the quadtree tree T_u as follows. If the maximum temperature saved at the current quadtree node x is lower than the threshold, we stop. Otherwise, we visit each child of x recursively. Assume y is a child quadtree node of x. Since the maximum temperature of quadtree node x is stored in any sensor in $B(p(x))$, where $p(x)$ is the parent node of x, obtaining the maximum temperature of node y requires a cost of walking from a sensor in $B(p(x))$ to a sensor $B(x)$. Therefore we can bound the cost of going from a quadtree node x to y by $O(L(u)/2^i)$, if x is i hops away from the root u in the logical quadtree. In other words, we define the weight of the quadtree edge from x to its children to be $O(L(u)/2^i)$. What remains is to bound the total weight of the quadtree edges that have to be traversed.

During the traversal of the tree, only the edges that are on the paths from hot sensors to the root are visited. We prune the tree T_u at the nodes with $\log(\sqrt{k(u)})$ hops away from u. We bound the weight of the traversed edges close to the root and the weight of the traversed edges far away from the root separately. Firstly, there are at most 4^i edges that are exactly i hops away from the root u, and their total weight is $4^i \cdot O(L(u)/2^i) = O(L(u)2^i)$. So the total weight of the traversed edges within $\log(\sqrt{k(u)})$ hops from the root u is bounded by the total weight of *all* the edges within $\log(\sqrt{k(u)})$ hops from u, which is no more than $O(L(u)\sqrt{k(u)})$. On the other hand, for the part of the tree with more than $\log(\sqrt{k(u)})$ hops away from the root u, the traversed edges are on at most $k(u)$ paths, each with total weight $O(L(u)/\sqrt{k(u)})$. Therefore, the weight of the edges far away from the root can be bounded by $k(u) \cdot O(L(u)/\sqrt{k(u)}) = O(L(u)\sqrt{k(u)})$. Thus the total communication cost is bounded by $O(L(u)\sqrt{k(u)})$. □

By Lemma 5.4, the cost of traversing all the canonical pieces is

$$\sum_{u\in C} L(u)\sqrt{k(u)},$$

where $\sum_{u\in C} L(u)^2 = ab$, $\sum_{u\in C} k(u) = k$. By the Cauchy-Schwartz inequality,

$$\begin{aligned}\textstyle\sum_{u\in C} L(u)\sqrt{k(u)} &\leq \sqrt{\textstyle\sum_{u\in C} L(u)^2 \sum_{u\in C}(\sqrt{k(u)})^2} \\ &= \sqrt{abk}.\end{aligned}$$

Therefore the total cost of the query is $O(D + \sqrt{abk} + (a + b)(\log(\min(a, b)) + 1)) = O(D + \sqrt{Ak} + P \log P)$. □

Clearly, the bound of the above algorithm applies to counting and isometric contour problems as well. However, in the above method, it is wasteful to traverse the whole "hot area" if what we care is just the number of hot nodes or the boundary of the hot area. Indeed, by slight modification of the above algorithm, we can obtain algorithms with query cost bounded by the number of hot nodes on the boundary rather than the number of hot nodes. For any given T, consider the nodes with reading at least T, and the unit-disk graph G_T spanned by those hot nodes. Each "hot island" is then a connected component of G_T. A node is on the boundary of a hot island if it is adjacent to some "cold" node, a node with temperature reading lower than T. The *perimeter* of a hot area is then the number of nodes on the boundary of the hot areas. In what follows, we show that for counting and isometric contour problems, the query cost of a slightly changed algorithm is only dependent on the perimeter of the hot area. Of course, in the worst case, the number of boundary nodes can be in the same order as the number of hot nodes. This happens when the hot area consists of many isolated small "islands", or when it is thin and long. However, for many scalar fields we encounter in practice, such as temperature field, we would expect that the boundary usually contains much fewer nodes than the hot area.

Corollary 5.5. *The cost of the counting query $(q, R, [T, \infty))$ is $O(D + \sqrt{As} + P \log P)$, where s is the number of sensors on the boundary of the set of sensors within R with temperature above T, D is the shortest path from the source to the query range R, and P and A are, respectively, the area and perimeter of R.*

PROOF. We save for a quadtree node u the number of sensors inside $B(u)$, as well as the maximum and minimum temperature. The only difference in the proof is in Lemma 5.4. We just need to count the hot sensors. When we traverse a tree T_u of a canonical piece u, if the current node x has a minimum temperature above the threshold, we output the number of sensors in $B(x)$ and don't recurse. We call such a node x a maximal node, because $p(x)$ must have a minimum temperature higher than T. So by the same argument as in Lemma 5.4, the communication cost of traversing the subtree T_u is therefore $O(L(u)\sqrt{s(u)})$, where $s(u)$ is number of maximal quadtree nodes inside $B(u)$. Since the temperature field is continuous, the collection of points with temperature beyond T are composed of a collection of regions corresponding to the maximal quadtree nodes. We also observed that

$$\sum_{u\in C} s(u) = O(s),$$

where s is the perimeter of the hot regions inside the range. So by the same argument in Theorem 5.3, the total cost of the counting query is bounded by $O(D+\sqrt{As}+P \log P)$. □

To answer the isometric contour query, we assume that the temperature field is approximated by interpolating the sample readings of the sensors in any standard way. For example, we construct a triangulation of the sensors. The temperature field within a triangle is calculated by interpolating the temperatures of its three vertices. For an isometric contour query (q, R, T), we denote by c the number of triangles intersected by the isometric contour inside R.

Corollary 5.6. *The cost of the query (q, R, T) reporting the isometric contour of temperature T inside R is $O(D + \sqrt{Ac} + P \log P)$, where c is the number of triangles intersected by the isometric contour inside R, D is the shortest path from the source to the query range R, and P and A are, respectively, the area and perimeter of R.*

PROOF. The same as the proof of Corollary 5.5. □

6. DISCUSSION

In this section, we discuss several variants of our algorithm. We also compare it with other schemes.

6.1 Reducing the counting cost

As shown in Theorem 4.1 and 5.3, our algorithm for visiting all the sensors almost matches the best possible lower bound when we require each sensor not to store too much information. However, for the counting problem, our algorithm has a multiplicative factor of $\sqrt{k}$ compared to the lower bound. The reason is that at each quadtree node, we store only the maximum of all the sensor values in the quadtree cell. Under such a scheme, we have to traverse to reach all the "hot" sensors (or "hot" areas) before we are sure how many hot sensors are contained in a quadtree cell. If we are able to improve the counting query for each quadtree cell u more efficiently, say in $O(L(u))$ steps, we can then improve our bound for counting to close to the lower bound.

Here, we sketch two ways to accomplish this goal. One method applies to the situation when there are not many different sensor values. In this case, we can use a distributed hash table to forward each sensor value to a particular sensor and then compute the number of sensors with sensor value above each possible value. Then, for each query, we simply direct the query to the sensor according to the hash value of the query. This suggests the approach in [15]. The other method applies when the sensor values are in an unknown range or with high accuracy. In this case, we sort the sensor values and store them on the sensors. The sorting can be done by using the distributed sorting algorithm on meshes [14] by taking advantage of the uniform distribution of the sensors. Then the query can be done by a binary search on the mesh. In both methods, we can achieve $O(L)$ query steps for a cell with side length L. This way, we can improve the counting cost to $O(D + P \log P)$, independent of k. This bound is close to the lower bound when the query range has small aspect ratio, in which case P is close to $\sqrt{A}$.

6.2 Answering temperature range queries

The storage scheme described in Subsection 5.1, where each quadtree node keeps the maximum/minimum temperature among its sensors, can be used to answer reporting queries for the "hot" sensors with temperature above T (or symmetrically, the "cold" sensors with temperature below T) with a total cost matching the lower bound. We notice that this is not true for either reporting or counting sensors within temperature range $[T_1, T_2]$. If we only keep the maximum/minimum temperature of the sensors on each quadtree node, it is possible that a quadtree node has a very high maximum temperature and a very low minimum temperature but no sensors inside the quadtree node fall in the temperature range $[T_1, T_2]$. This problem arises due to the discrete sampling and sparse deployment of the sensors. In fact, for range searching in continuous temperature field, if the temperature range of a quadtree node overlaps the query range $[T_1, T_2]$, the query is not empty and therefore the cost of the traversal of the quadtree node can be justified accordingly. This is no longer true for a discrete sensor field. Therefore, to answer queries on temperature ranges we should have each quadtree node store a summarized distribution of its sensor readings such that more effective pruning can be made during the traversal of the quadtree.

6.3 Reducing the storage

The algorithm we described requires $O(\log n)$ values stored at each sensor or $O(n \log n)$ storage overall. This is because each sensor stores $O(\log n)$ maximum values, one for each quadtree cell it is in. The advantage of such a scheme is that it avoids assigning a leader to each quadtree cell and forcing each query to communicate with the leader. This way, we can achieve better fault tolerance and load balance, and simplicity of the query algorithm. However, in the cases when the memory or storage size on a sensor is a more important factor, we may switch back to the scheme that assigns leaders to each quadtree cell, i.e., for a quadtree node u, we save the maximum temperature $t(u)$ at the centers of itself and its sibling quadtree nodes. See Figure 4 for an example. This is done in the same way as the geographic hash table (GHT) [21], except that the hashed locations are not random. Routing to the locations of the leaders from an arbitrary sensor could be done by geographic routing since the locations can be calculated from the quadtree node they represent. Since the value $t(u)$ is stored in a constant number of sensors, the total storage necessary is $O(n)$. On average each sensor stores only $O(1)$ values. The query cost in the memory efficient model is still bounded by Theorem 5.3.

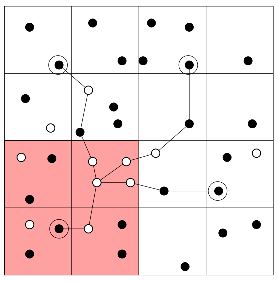

Figure 4. Memory efficient storage scheme: the maximum temperature of the pink quadtree node is hashed to the centers of itself and its sibling quadtree nodes.

6.4 Pruning the query range

The cost of a range search query, as bounded in Theorem 5.3, is the worst case cost. In fact, as the source q sends out a message toward the query range by geographic forwarding, the intermediate sensors on the routing path can start to try to prune the query range based on the values stored on them. For example, if the maximum value $t(u)$ of a quadtree cell u stored at sensor p is below the query threshold T, and $B(u)$ covers the query range R completely, then there cannot be any hot sensors inside R. Sensor p can thus answer the query by responding a message of "no hot sensors" to the source q. If $B(u)$ covers only part of the query range, then $B(u) \bigcap R$ can be chopped off, and the

	Flooding	Global Warehouses	Fractional Cascading
Max storage	$O(1)$	$O(n)$	$O(\log n)$
Query cost	$D + A$	$O(\sqrt{n/w})$	$D + O(\sqrt{A}k + P \log P)$
Update cost	$O(1)$	$O(wn\sqrt{n})$	$O(n \log n)$

Figure 5. Comparison of the performance of three information storage/retrieval schemes: flooding, global warehouses and fractionally cascaded information.

query range is reduced to $R \setminus B(u)$. Such pruning of the query range can continue until the (updated) query range is reached. Since the sensors closer to the query range R have more detailed information about R, we can imagine that as the query gets closer and closer to the query range, the pruning becomes more and more effective. Although such heuristics will not improve the worst case performance, as shown by our lower bound results, it may help reduce the query cost on average.

6.5 Handling dynamic updates

In all the previous discussion, we did not mention how dynamic updates are done. Since the sensor values are constantly changing, it is important for the information aggregation algorithm to deal with such changes efficiently. In our algorithm, when the value of a sensor changes, we need to propagate the change up the quadtree. The propagation stops when the maximum value of a quadtree node no longer changes. In the worst case, the update can go up all the way to the root. However, we expect the overall amortized cost to be low. In our original scheme, the change of a maximum value needs to propagate to all the sensors in the quadtree node and its siblings. It may take some time for the propagation to finish before a new query arrives. One solution is to associate a time stamp with each maximum value stored in each sensor. When a query reaches a sensor, if the time stamp indicates the value is out of date, then the query will be routed toward the leaders of the quadtree node until a "fresh" value is encountered.

6.6 Comparison

In Section 3, we informally described the performance of the flooding and global warehouse methods. Here, we provide a quantitative comparison between the fractional cascading approach with those two methods in terms of storage, query cost and the total amount of information propagated per update. Assume the query range has area A and perimeter P, and k is the number of hot sensors inside the query range R.

Flooding. The source q sends a message to the query region and floods all the sensors inside the query region to collect the answer.

Global warehouses. In this setting w sensors are selected to save the information about the temperature readings of all the sensors. For example, each of the warehouses saves a complete quadtree structure. The best way to place the global warehouses is to put them uniformly so that any sensor has a warehouse within distance $O(L/\sqrt{w})$, if the sensors are located in a box of side length L, $L \sim \sqrt{n}$. Notice that there's no aggregation during the transmission of information to the warehouses. Each sensor sends $O(1)$ bits of information to every warehouse. When the sensors have new readings, the total amount of information transferred during an update scheme is $O(Lnw) = O(wn\sqrt{n})$ in the worst case.

Figure 5 summarizes the maximum storage, query cost, and update cost for these three methods. We can see that the flooding method works better when the query range is small, while the warehouse method provides better worst case complexity with an appropriately chosen w. Our method uses much less storage compared with the global warehouse method. Compared with the flooding method, our method is less dependent on the query range size.

7. CONCLUSION

In this paper, we proposed a quadtree based algorithm for fractionally cascading information in sensor networks. Our method allows efficient range query in sensor networks. We provide rigorous analysis of the performance of our algorithm and show that it is close to the optimum under a reasonable cost model.

This paper studied the worst-case performance bound of the aggregation and query scheme in a sensor network. It will be interesting future work to study the average case performance bound or probabilistic performance.

Acknowledgements:

Research by Jie Gao and Leonidas Guibas was supported by the NSF grant CCR-0204486, the DARPA Sensor Information Technology Program under contract F30602-00-C-0139, ONR MURI grant N00014-02-1-0720, and a grant from the Stanford Networking Research Center. Jie Gao is also supported by an IBM Ph.D. fellowship.

8. REFERENCES

[1] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[2] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(3):133–162, 1986.

[3] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986.

[4] F. Dehne, A. Ferreira, and A. Rau-Chaplin. Parallel fractional cascading on hypercube multiprocessors. *Comput. Geom. Theory Appl.*, 2(3):141–167, 1992.

[5] L. Doherty, B. Warneke, B. Boser, and K. Pister. Energy and performance considerations for smart dust. In *International Journal of Parallel Distributed Systems and Networks*, pages 121–133, 2001.

[6] A. Faradjian, J. E. Gehrke, and P. Bonnet. GADT: A Probability Space ADT for Representing and Querying the Physical World. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, San Jose, California, February 2002.

[7] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heideman. An implementation of multi-resolution search and storage in resource-constrained sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.

[8] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database. Technical Report CS-02-771, University of Southern California, 2002.

[9] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, Alaska, May 2003.

[10] IEEE. Draft standard for part 15.4: Wireless medium access control and physical layer specifications for low rate wireless personal area networks (lr-wpans). 2002.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *ACM Conf. on Mobile Computing and Networking (MobiCom)*, pages 56–67, 2000.

[12] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proc. 6th Annual ACM Mobile Computing and Networking (MobiCom '00)*, pages 243–254, 2000.

[13] D. Kempe, J. M. Kleinberg, and A. J. Demers. Spatial gossip and resource location protocols. In *ACM Symposium on Theory of Computing*, pages 163–172, 2001.

[14] T. Leighton. *Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.

[15] X. Li, Y.-J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.

[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sesnor Networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, Massachusetts, December 2002.

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 491–502. ACM Press, 2003.

[19] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215–241, 1990.

[20] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing in mobile ad hoc networks. In *ICDCS Workshop on Wireless Networks and Mobile Computing*, pages D71–D78, 2000.

[21] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets, 2002.

[22] R. Tamassia and J. S. Vitter. Optimal cooperative search in fractional cascaded data structures. *Algorithmica*, 15(2), 1996.

[23] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proc. 21st Int. Ann. Joint Conf. IEEE Comput. Comm. Soc. (INFOCOM)*, pages 3–12, 2002.