Optimization of In-Network Data Reduction

Joseph M. Hellerstein*† Wei Wang*

*UC Berkeley and †Intel Research Berkeley

{jmh,wangwei}@eecs.berkeley.edu

Abstract

We consider the in-network computation of approximate "big picture" summaries in bandwidth-constrained sensor networks. First we review early work on computing the Haar wavelet decomposition as a User-Defined Aggregate in a sensor query engine. We argue that this technique can be significantly improved by choosing a function-specific network topology. We generalize this discussion to a loose definition of a 2-level optimization problem that maps from a function to what we call a *support graph* for the function, and from there to an aggregation tree that is chosen from possible subgraphs of the physical network connectivity. This work is frankly quite preliminary: we raise a number of questions but provide relatively few answers. The intent of the paper is to lay groundwork for discussion and further research.

1 Introduction

Wireless sensor networks must operate with significant constraints on energy and bandwidth consumption. This presents challenges for interactive analysis of data in sensornets, since data analysts tend to desire a big-picture view of the data before "drilling down" to specific queries. The big-picture queries can range over all the data in the network, but fortunately approximate answers are often sufficient for these purposes. Techniques to provide approximate answers to resource-intensive queries of this sort were explored by a variety of researchers in traditional database scenarios (e.g., [6, 8]).

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
`http://db.cs.pitt.edu/dmsn04/`

In this paper we explore some initial ideas and challenges in performing online, in-network data reduction in sensor networks. Data reduction techniques can be used to provide synopses or "sketches" that can be used to approximately answer queries. Our main contribution here is not to present specific results, but to rough out a set of ideas and research challenges that we hope the community can explore and define further.

We begin by describing in some detail two techniques for in-network computation of Haar Wavelets. We hinge this discussion on the Haar *support tree*, a logical dataflow specification that describes the ordering constraints on combining data values. We show that an earlier idea for in-network computation of the Haar does not observe the constraints of the support tree, and instead produces biased results. We then consider constraining the network topology to generate a physical *communication tree* that observes the constraints of the logical Haar support tree. We present the surprising observation that a correct communication pattern for the Haar support tree results in a *binomial communication tree* at the network layer. This insight leads to some relatively crisp questions surrounding the optimization of communication topologies for computing Haar wavelets in-network.

Given this specific example as background, we pose a more generic (albeit vaguely defined) family of optimization problems for doing in-network data reduction, by focusing on the general problem of mapping from support graphs to communication graphs for various computations. We also raise various challenges in transferring this algorithmic work to practice.

2 Case Study: Wavelets

Wavelets have been widely used in the database literature as a data reduction technique (a tutorial is presented

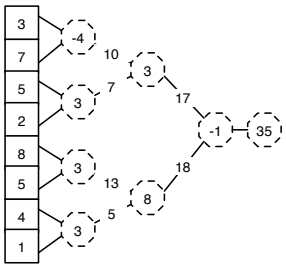


Figure 1: A column of a table and its Haar wavelet support tree (sometimes called an "error tree"). The output of the wavelet transform in this example is [35, -1, 3, 8, -4, 3, 3, 3].

in [11]). Aggregate queries can be answered approximately by running them over compressed wavelets of a raw dataset. Wavelets have a number of attractive properties, including their mathematical simplicity, and their ability to provide "multi-resolution" results by incrementally fetching more of the wavelet from a disk or network.

2.1 A Brief Primer on Haar Wavelets

The Haar wavelet is the simplest and most popular example of the wavelet family. The Haar is also easy to explain; we give a brief sketch here. Given an array of numbers (e.g., one column of a database table), it pairs up the neighboring numbers in odd and even positions (e.g. rows of the table), and transforms them into two different numbers: their sum and their difference. The differences are stored, and the sums are passed into a recursive application of the procedure. The recursion can be visualized as a tree, as in Figure 1[1]. The numbers ("coefficients") stored at each internal node in the tree represent the differences between the overall sum of leaves in the left and right subtrees of the node; the edges are labeled with the sums that are passed up. The root represents the sum of all the entries in the original array. We call this tree the *support tree* of the Haar wavelet: edges in the tree represent data dependencies, where each internal node is computed as a function of its children, and the leaves underneath a node represent the support of the value in that node. The output of the Haar transform can be produced by a breadth-first traversal of the (non-leaf) nodes of the support tree, though in practice there are coding algorithms that do not require constructing and traversing such a tree [16].

The decoding of the transformed data can be done in a straightforward fashion starting from the root and recursing downwards: given the overall sum s at the root, and the difference d at the node below, the overall sums of the left and right subtrees are calculated as $(s+d)/2$ and $(s-d)/2$ respectively, and the process can then recurse to the leaves.

As described, the output of the Haar transform is exactly the same size as the input. However, a simple scheme can be used to lossily compress the wavelet by truncating the list of coefficients. The basic idea is to only keep coefficients with high absolute values[2], and "round" the remaining coefficients to zero. In our example of Figure 1, truncating to the top 3 coefficients gives [35, 0, 0, 8, -4, 0, 0 ,0]. The resulting output array has mostly zero-valued entries, and can be represented compactly via a number of well-known techniques (e.g., via $(position, value)$ pairs for the non-zero entries, or run-length encoding.) Decoding our truncated example wavelet reconstructs the input as [2, 6.75, 4.375, 4.375, 6.125, 6.125, 2.125, 2.125]. Note that wherever a node in the support tree was rounded to zero, the reconstructed leaves in the corresponding subtree moved closer together in value. Dropping coefficients "smooths" differences in the original data.

If the full wavelet encoding is available somewhere – e.g. on a disk, or across a network – then the number of "unrounded" coefficients fetched locally can be increased incrementally in a "multi-resolution" manner, to remove these smoothing effects. Each new coefficient fixes a more subtle smoothing than the previous. This incremental improvement in the reconstruction is one attractive feature of wavelets.

A final side-note is merited regarding the treatment of set-valued data like columns of database tables. Wavelets are a sequence-encoding scheme, preserving the ordering of values in the input. In databases, this input ordering is arbitrary by definition. Given that any ordering is acceptable, an open question is to choose an ordering of the input data for which a wavelet truncated to the top k coefficients is most effective. For numeric data, sorting the table is a natural option; an extension of this idea for inte-

[1]The example builds a 1-d wavelet. Multi-d wavelets are analogously built with trees of fan-in 2^d.

[2]Typically the values are normalized by dividing by $\sqrt{2^i}$ where i is the height of the node above the leaves. Normalization does not affect the examples or algorithms here.

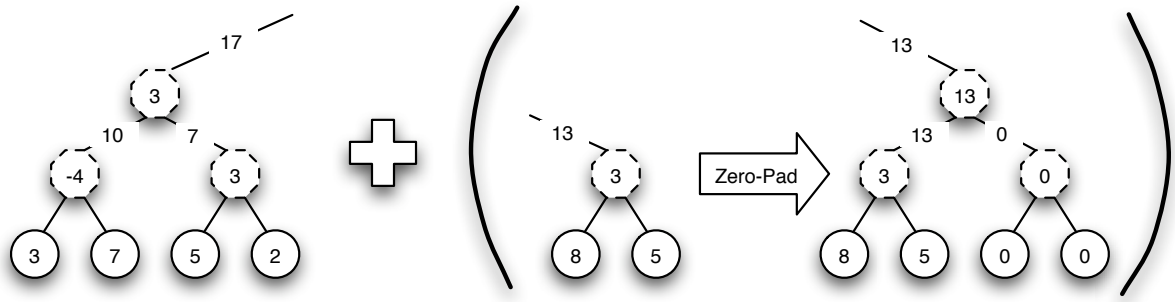


Figure 2: Given support subtrees of differing sizes, the PM technique zero-pads the smaller subtree before combining them.

ger data is the Wavelet Histogram, which run-length encodes the sorted column into (value, frequency) pairs and performs a wavelet transform on the resulting sorted frequencies [13]. For categorial attributes, the best choice of sort-order is an open question; it is likely to be dependent on the wavelet basis functions chosen (e.g. Haar, Daubechies-4, Mexican Hat, etc.)

2.2 Haar Wavelets as a Distributed UDA

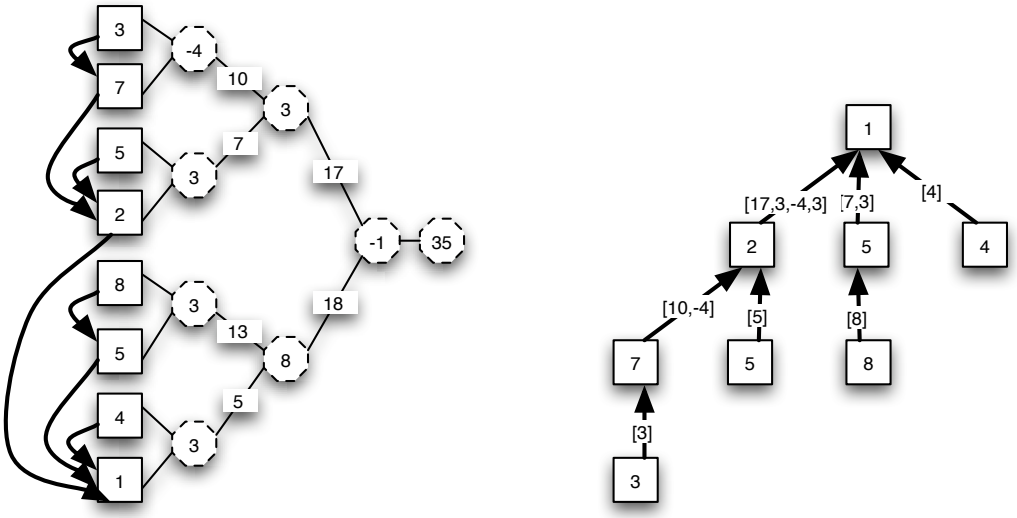
Earlier work based on the TinyDB system presented a User-Defined Aggregation (UDA) technique to compute a Haar wavelet over readings gathered in a sensor network [10]. We refer to this as the Pad-Merge or PM technique, and briefly review it here.

As in extensible databases, UDAs in TinyDB are represented by a triplet of functions: a merging function f, an initializer i, and an evaluator e. The initializer converts a scalar input value into an opaque *partial state record (PSR)*, the merging function takes two PSRs and combines them into a new PSR, and the evaluator takes a PSR and produces an output scalar value. In sensor-net query systems like TinyDB, an aggregation query is disseminated to participating sensor nodes, which call the initializer function on their local reading and then communicate PSRs up a *communication tree* of network links to the query node. When a node N receives a PSR from a child in the tree, it calls the merging function to merge the incoming PSR into N's current PSR; when N has merged in all of its children's PSRs, N sends the merged PSR to its own parent. Details of this aggregation scheme, including the dissemination of queries and construction of communication trees, can be found in the literature [12].

The PM technique uses a distributed, bottom-up scheme to construct a Haar support tree like that of Figure 1. It has a total communication cost that is linear in the number of nodes of the network (one fixed-size message per node). The PSRs in the PM technique are essentially arrays of k wavelet coefficients represented as $(position, value)$ pairs. Each PSR corresponds to a subtree of a complete Haar support tree. The main logic in the PM technique is in the merging function, which takes two arrays of wavelet coefficients (representing two Haar subtrees), generates a new set of wavelet coefficients representing the two trees connected by a new root, and keeps the top k of those coefficients as the new PSR[3]. Upon completion, the PM technique produces k large wavelet coefficients that can be used to lossily reconstruct the input data.

A Haar support tree is a balanced binary tree. But aggregation in TinyDB imposes no structure on the communication tree, and hence it does not control the order in which PSRs are merged. The merging function can be invoked on two *arbitrary* PSRs, which may represent Haar subtrees of differing heights. To handle this, the PM approach proposes a zero-padding technique to "promote" the smaller of the two input PSRs to a tree of the same height as the larger: it pads the smaller PSR with an appropriate number of zero-valued leaves until it becomes

[3]The order in which PSRs are combined recursively determines the left-to-right ordering of the leaves of the Haar tree. In our discussion here we focus on set-oriented query scenarios where this order – or, equivalently, IDs of the nodes – is insignificant. Preserving the order or node IDs can be done in a number of different ways that would complicate our discussion here unnecessarily.



[t]

Figure 3: An in-network computation of the Haar wavelet of Figure 1. The left side annotates the (logical) support tree with dark arrows representing physical message-passing between the sensor nodes. The right side of the figure shows just the (physical) communication tree, i.e., the leaf level of the left side. Each edge on the right is labeled with the wavelet coefficients sent.

a balanced binary tree of the same height as the larger PSR (Figure 2). This guarantees that the PM technique always merges two PSRs of the same size, and hence always constructs balanced binary Haar support trees.

If the PM technique never truncates any coefficients, it can reconstruct the data perfectly: the extra zeros introduced by padding can be correctly accounted for and deleted in the decoding process. However, in the practical cases where the PSR is much smaller than the number of nodes in the network, each merging step has to truncate to the top k coefficients. When zero-padding is used, the truncating can smooth the spurious zeros across the true data. In the end, the PM technique will produce a k-coefficient wavelet that is not as accurate as the one that would be produced in a centralized implementation of the Haar encoding – the PM wavelet will incorrectly bias the reconstructed data toward zero, in many cases in a significant way.

2.3 Haar-Specific Network Topologies

The PM technique introduces bias when padding Haar support subtrees of unequal size. Imagine that one could guarantee that only equal-sized subtrees were merged. Then no zero-padding would be needed, and the correct top-k wavelet coefficients would be produced as a result. In this section we explore the possibility of achieving such an invariant by controlling the sensor network topology used for aggregation in the network.

For purposes of illustration, assume for a moment that we have a *fully-connected* communication network with nodes numbered 1 through 2^l. Our goal is to construct the Haar support tree bottom-up by passing messages between nodes. By convention, we will assume that lower-numbered nodes will pass messages to higher-numbered nodes. The process begins at the leaves of the support tree: node 1 passes its value to node 2, node 3 passes its value to node 4, etc. The even-numbered recipients pass along PSRs that contain their top k difference coefficients as well as their sum: node 2 passes its PSR to node 4, node 6 passes its PSR to node 8, etc. At the end of this process, the contents of the Haar support tree would be distributed throughout the network, with the top-k coefficients and the overall sum residing at node 2^l. This communication pattern is depicted by the directed arrows in the left side of Figure 3.

Given our assumption of a fully connected sensor network graph, this distributed algorithm employs a very stylized subgraph that comes from the data structures literature: the *binomial tree* [4] (right hand side of Fig-

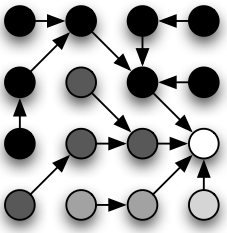


Figure 5: A binomial tree embedded in a radius-1 grid.

ure 3). In a binomial tree of 2^l nodes, the root has l children, which are binomial trees of 2^i nodes for $i \in 0, \ldots, l-1$. The depth and maximum fan-in of a binomial tree are both logarithmic in the number of nodes.

We can now relax our unrealistic requirement of full connectivity in the sensor network, and ask whether this technique is feasible in practice. This reduces to two basic questions: (1) do binomial trees naturally occur as subgraphs of practical sensornet communication graphs, and if so, then (2) can an efficient, distributed topology-selection algorithm be devised to find and maintain a binomial subtree topology in a sensor network?

It would be interesting to study this question empirically, and/or to analyze it formally for random graphs from typical distributions. Here we simply provide a bit of intuition from the canonical simplistic sensornet model of an equally-spaced 2-d grid of nodes with communication radius of 1 grid-square per node. In a 4×4 grid, it is certainly possible to find binomial trees (Figure 5). Note however that in two dimensions each node has only 8 neighbors, and the root of a binomial tree of size 2^l has l children. Hence clearly any 2-d grid topology of more than 256 nodes will not have a binomial tree embedding unless its communication radius is greater than 1. Similarly, since the corner of a grid has only 3 neighbors, there is no binomial tree rooted at a corner of our 4×4 grid of Figure 5.

3 Generalizing the Haar Example

Haar wavelets are only one of many non-trivial aggregation functions that may be of use in sensor networks. The discussion above illustrates a number of interesting, general problems that arise in computing such complex aggregates efficiently. In this section we briefly sketch a set of research problems that arise in this space.

3.1 A Static Optimization Problem

Section 2.3 raises the challenge of finding communication trees that match the Haar wavelet support tree. This is an example of a more general optimization problem in sensornet aggregation. The challenge is to take any aggregation function and map it onto the graph of radio connectivity in the network. This can be viewed as a multi-layer optimization problem: as illustrated in Figure 4: (a) a support graph must be chosen for the aggregation function, and (b) the support graph must be mapped onto a communication tree; the communication tree in turn is constrained to be a subgraph of (c) the radio connectivity graph of the sensornet. Note that depending on the aggregation function, there may be more than one satisfying support graph for step (a). Similarly, in step (b) there are multiple communication trees corresponding to a chosen support graph, more than one of which may be a subgraph of the radio connectivity.

In the case of the Haar wavelet, the mapping from support graph to communication graph was quite elegant: a balanced binary support tree became a binomial communication tree. Since the properties of binomial trees are well known, they are amenable to analysis and (hopefully) simple construction and maintenance algorithms. It is unclear whether the mappings of other support graphs into communication graphs will be as elegant. The curious reader is encouraged to play with the Daubechies-4 wavelet as a more complex example, since it has a support DAG rather than a support tree. The general mapping problem itself is of interest, as is the question of characterizing the communication graphs at the output.

As noted in the previous section, it may in some cases be impossible to find a communication graph in the network to match a particular support graph for a function. In such cases, two options are available. One is to achieve such a topology as an *overlay* network, by having some sensors *forward* PSR messages directly to other nodes without applying the merging function. This of course causes overheads that spoil the ideal linear communication cost of many aggregates. The second option is to always apply the merging function on arriving PSRs regardless of data dependencies in the support graph; logically this reshapes the support graph that gets computed. This is exactly the approach taken by the PM technique for Haar wavelets. Ideally this latter approach

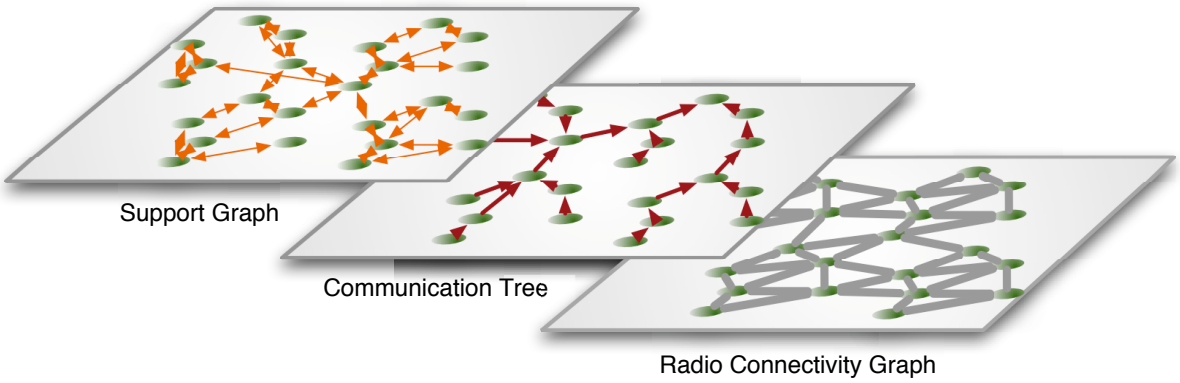


Figure 4: The general optimization problem needs to choose a Support Graph, and map it to a Communication Graph that is a subgraph of the Radio Connectivity Graph.

should include a technique to quantify the error introduced by such inappropriate merging.

The general optimization problem is as follows. Given an aggregation function, a connectivity graph, and a cost function to minimize, the challenge is to choose a min-cost communication graph in the network that is a subgraph of the connectivity graph. The communication graph must be annotated to differentiate between cases of PSR forwarding and PSR merging. The cost function is likely to be a multi-objective metric, incorporating perhaps such issues as bandwidth, latency, power consumption, and bounds on errors in the result.

3.2 Real-World Complications

This optimization problem is relatively well-defined, but not entirely realistic. Here we highlight additional challenges that are likely to arise in practice.

The first is the very real issue of packet loss in sensor networks. Loss probabilities on radio links can be estimated, and added as inputs to the optimization problem. But this leaves the question of how to deal with loss. A natural option is to implement network retries; the expected number and cost of retries can be translated in the cost metric to bandwidth, latency and power consumption. A second option is simply to tolerate loss, and estimate the loss in accuracy of the answer. A third, intriguing direction is the use of forward error correction. Naive application of error-correcting codes seems like a bad idea, since the codes are traditionally used to preserve opaque packets. Given our knowledge of application semantics, it is interesting to explore the joint design of error-correcting aggregation functions. The recent work on duplicate-insensitive distinct count sketching [3] may seen as an example of this idea. A generic challenge with any of these schemes is to minimize waking time: if a node chooses not to propagate any data (e.g., because its coefficients are below a threshhold) it should be able to power down. This is complicated by the problem of loss, since it is unclear how receivers differentiate between lost packets and unsent packets.

A second critical challenge is that of network dynamism. Experience shows that connectivity in a sensornet changes over time as a function of many factors. Given that the physical graph will change over time, a dynamic reoptimization technique is needed for the problems sketched above, and preferably one that works in a distributed fashion with minimal communication requirements.

An additional, fundamental challenge arises at the architectural level. This paper advocates algorithmic optimizations that collapse traditional boundaries between application-level logic and various parts of the network stack (e.g. topology construction, loss handling, etc.) This raises the challenge of architecting a system that allows users defining new aggregation functions to describe acceptable networking choices with a minimum of fuss. This is an extensibility interface that is not well understood. A better understanding of this interface might also provide guidance in choosing data reduction functions to compute. For example, the support graphs of various wavelet variants (Haar, Daubechies-4, etc.) are

quite different. Understanding how to *describe* these differences compactly to a system might also provide analytical insight into their relative merits in terms of mappability to communication graphs.

Finally, this discussion raises the question of what one does with multiple concurrent functions with competing desires – e.g. a query that requests the simultaneous computation of two very different aggregates.

4 Open Issues and Alternatives

This paper describes a relatively focused family of optimization challenges. In this section we briefly touch on some broader issues and alternative approaches.

An important challenge in this context is to handle changes in the data while the aggregation protocol is running. Multi-resolution schemes like wavelets can let users watch detail accumulate as coefficients are passed up in multiple rounds of communication, in the spirit of Online Aggregation [9]. However, during the multiple rounds of communication, the data itself may be changing, and it may be more beneficial to send newer, coarse-grained data rather than increasing refinements on stale data. In this vein, it might be beneficial look at spatio-temporal wavelet encoding, and consider which coefficients of the spatio-temporal wavelet to communicate at each timestep. This tradeoff emcompasses data properties and user desires, and it inherently a mix of systems, coding, and HCI issues.

The traditional database approach to aggregation has a unidirectional dataflow that results in the one-way communication trees we have discussed here. A broad class of data analysis techniques can be more efficiently computed in two communication rounds: one up a tree and the other back down. This includes multi-dimensional regression, Fast Fourier Transforms, and Bayesian belief propagation, all of which can be computed via the Junction Tree algorithm [1]. These techniques have been mapped into the sensornet domain in recent years [7, 15]. But current sensornet query engines have yet to incorporate these approaches into their architectures or languages, and the integration may require a new architecture beyond analogies to Object-Relational UDAs. It is worth noting that many of the problems suggested here are related to work being studied in the Junction Tree context [15].

Another fruitful vein of exploration is to design data reduction techniques whose merging function is fully commutative and associative. The network optimization for these aggregates is therefore unconstrained by the choice of support tree. AMS sketches [2] are one example that may be a good alternative to wavelets. Nath and Gibbons propose a scheme to additionally introduce duplicate insensitivity to aggregates in a general way [14]. Duplicate insensitivity removes the constraint of the communication graph being a tree, allowing for arbitrary "diffusion" or "gossip" of messages.

Wavelets have been proposed for sensor networks in the work of Ganesan, et al. on DIMENSIONS [5]. DIMENSIONS does not perform any distributed wavelet computation. Instead it has two main components: (a) it uses local wavelets to lossily compress archival storage of readings over time at each node in the network, and (b) it embeds a geographic quad-tree in the network to provide distributed, hierarchical spatial summarization. Each node of the quad tree receives the (wavelet-encoded) data from the nodes below, decodes it to form a 2-d array, and re-encodes the array into (threshholded) 2-d wavelet coefficients used both for lossy local storage and for communication further up the quad-tree. DIMENSIONS blends two approaches to hierarchical data reduction: local wavelets and distributed quad trees. An interesting question is whether a distributed multi-dimensional wavelet of the form described in this paper could be extended appropriately to achieve the functionality of DIMENSIONS in a unified fashion.

5 Conclusion

If sensornet query engines are to succeed, they need to either provide a wide range of useful built-in functionality, or be easily extended to incorporate new functionalities. Given the relative immaturity of the area, it is unlikely that we will anticipate many of the important features in advance. The traditional User-Defined Aggregation functionality of extensible databases should be a key feature in sensornet query systems, and optimization of UDAs over networks will be a key challenge. Perhaps the most critical aspect of the work described here is architectural challenge raised: how do users define the merging rules for complex UDAs to the system, and are there general optimization techniques to take such rules and use them to achieve good performance?

Acknowledgments

Thanks for conversation and feedback to Amol Deshpande, Christos Faloutsos, Minos Garofalakis, Phil Gibbons, Carlos Guestrin, Sam Madden, Yossi Matias, Mark Paskin, Kannan Ramchandran, and Mehul Shah. Mark Paskin devised the visualization of the layered optimization problem for his work on distributed inference [15].

References

[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Trans. Info. Theory*, 46(2), 2000.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 20–29, Philadelphia, PA, 1996.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. International Conference on Data Engineering (ICDE)*, Mar. 2004.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[5] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proc. First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[6] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. M. andf S. Muthukrishnan, S. Ramaswamy, and T. Suel. AQUA: System and techniques for approximate query answering. Technical report, Bell Laboratories, Murray Hill, NJ, Feb. 1998.

[7] C. Guestrin, R. Thibaux, P. Bodik, M. A. Paskin, and S. Madden. Distributed regression: An efficient framework for modeling sensor network data. In *Proc. 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.

[8] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis with CONTROL. *IEEE Computer*, 32(8), August 1999.

[9] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *Proceedings of the ACM SIGMOD*, pages 171–182, Tucson, AZ, May 1997.

[10] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *2nd International Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.

[11] D. Keim and M. Heczko. Wavelets and their applications in databases. In *Proc. International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Symp. Operating Systems Design and Implementation (OSDI)*, 2002.

[13] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, pages 448–459, Seattle, Washington, 1998.

[14] S. Nath and P. B. Gibbons. Synopsis diffusion for robust aggregation in sensor networks. Technical Report IRP-TR-03-08, Intel Research, 2003.

[15] M. A. Paskin and C. E. Guestrin. A robust architecture for distributed inference in sensor networks. Technical Report IRB-TR-03-039, Intel Research, 2003. Submitted for publication.

[16] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.