Adaptive and Decentralized Operator Placement for In-Network Query Processing

Boris Jan Bonfils and Philippe Bonnet

Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen, Denmark
bbo@bording.dk, bonnet@diku.dk
http://www.distlab.dk/vaquita/

Abstract. In-network query processing is critical for reducing network traffic when accessing and manipulating sensor data. It requires placing a tree of query operators such as filters and aggregations but also correlations onto sensor nodes in order to minimize the amount of data transmitted in the network. In this paper, we show that this problem is a variant of the task assignment problem for which polynomial algorithms have been developed. These algorithms are however centralized and cannot be used in a sensor network. We describe an adaptive and decentralized algorithm that progressively refines the placement of operators by walking through neighbor nodes. Simulation results illustrate the potential benefits of our approach. They also show that our placement strategy can achieve near optimal placement onto various graph topologies despite the risks of local minima.

1 Introduction

Sensor networks are a promising platform for a new generation of monitoring applications [4]. In the recent years, research has shown that clusters of densely deployed sensor nodes arranged in a multi-hop network allow for improved sensing (via collaborative signal processing [17]) and improved energy efficiency [14]. Because data transmission is orders of magnitude more costly than processing on a sensor node [14], data processing should be pushed inside the sensor network whenever it reduces the amount of data to be transmitted [7]. The term *in-network processing* has been tossed to denote data processing that takes place inside the network [6].

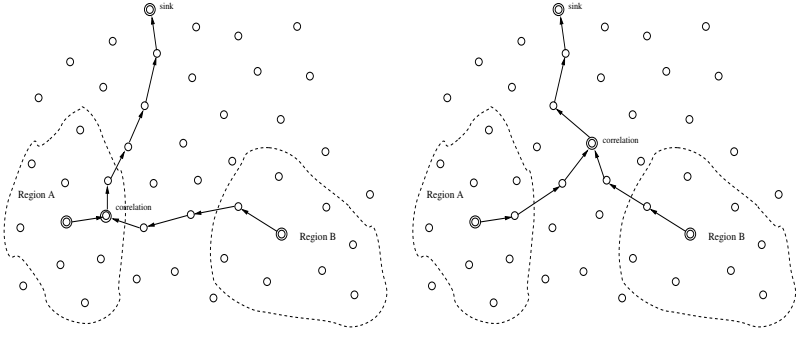
It is now generally admitted that access to data in a sensor network should be declarative [5]: Users formulate queries to access the data they are interested in. The exact definition of these queries is an open issue. We take a database perspective and consider that a query is a tree of *operators* that aggregate, correlate or filter data streams [2,3,16]. In this paper we tackle the issue of operator placement for in-network query processing, i.e., on which sensor nodes should query operators be placed?

We give an example to illustrate the importance of operator placement. A user is correlating detections obtained from two distinct regions (in the context

F. Zhao and L. Guibas (Eds.): IPSN 2003, LNCS 2634, pp. 47–62, 2003.

of animal monitoring or vehicle observation): The user should be notified whenever similar targets are detected in the two distinct regions within a given time window. For the sake of simplicity we consider that one node generates detections in each region[1]. A correlation operator is pushed inside the network. This operator takes as input the detections from the two regions and generates an output whenever a similar target has been detected in the two regions within a given time window. A gateway node is the sink that consumes the data generated by the correlation operator.

Let us now discuss the ideal placement for the correlation operator, i.e. the operator placement that minimizes the amount of data transmitted in the network. Consider that initially both regions produce detections that are not correlated and that one region produces more detections than the other. The correlation operator is very reductive (it does not output data). Intuitively, its optimal placement is on the shortest path between the two nodes that generate detections, and closer to the node that produce more data (see Figure 1(a)). The exact position of the correlation operator depends (i) on the rate at which data is produced by the operator and both sources, as well as (ii) on the path length between the sources, the operators and the sink. In a second configuration, the detections are somewhat correlated. As a result, the correlation operator produces more data and its optimal placement is closer to the sink (see Figure 1(b)).



(a) Inequal number of detections in each region – Low correlation

(b) Equal number of detections in each region – Low correlation

Fig. 1. Examples of Operator Placements

The optimal placement of the correlation operator corresponds to the solution of a task assignment problem: The problem is to find the mapping of operators to sensor nodes that minimizes the amount of data transferred over the network.

[1] Possibly, a collaborative signal processing algorithm generates detections and elects a representative node in each region. At different points in time, different nodes might be elected to produce the detections.

We give a more formal problem definition in Section 2. Even though polynomial algorithms exists for some versions of the task assignment problem, they cannot be used for our purpose because:

1. We do not assume global knowledge about the sensor network. The algorithms that solve the task assignment problem are centralized and require complete topological information [1]. It would be expensive in terms of data transfer and quite inaccurate for a central site to maintain this information. In addition, the amount of information maintained on each node should be minimal because of memory limitation, In order to limit maintenance overhead, this information should be local - each node should only maintain information about close-by nodes. As a result, we need to devise a *decentralized* solution where each node maintains local information.
2. The placement of an operator needs to be recomputed as the conditions in the network change. In our example, the optimal placement of the correlation operator changes as the correlation between the detection evolves, but also as the number of detections produced in each region changes. It would be inefficient to place an operator on a sensor node once and for all. In our example, if the correlation operator is permanently placed at the sink node while the detections from both regions are not correlated, then data is transmitted all the way to the sink for no result. It would have been more efficient to compute the correlation as close to the sources as possible. Our solution must thus be *adaptive*.

In this paper, we propose a decentralized and adaptive solution to the operator placement problem. In our solution, sensor nodes continously refine the placement of operators in order to minimize the amount of data transferred over the network. While one node is *active* executing an operator, a set of candidate nodes, that we denote *tentative* nodes, estimate the cost of running this operator (the cost is a function of the amount of data received and produced by a node). At regular intervals estimated costs are compared with the actual cost measured on the active node and execution is transferred to the node with the lowest cost.

We first give a formal problem definition and then we detail our contributions:

- We define *neighbor exploration*: a decentralized exploration strategy that continously refine the placement of operators towards the nodes with minimal costs.
- We describe a decentralized and adaptive algorithm that implements the neighbor exploration strategy based on the notions of active and tentative nodes.
- We present simultation results that illustrate the potential benefits of our approach.

Our simulation results show that our approach is viable. However, they do not allow us to lead a complete study. A key question that we do not tackle in this paper concerns the overhead generated by our decentralized algorithm - in terms of messages exchanged. Another interesting question concerns the reactivity of

our approach (how fast it can adapt to changing conditions – e.g., data rate, data correlation). In order to study these issues, we have implemented our solution on top of directed diffusion [6]. We are currently running experiments using NS. Our initial results are very promising.

2 Problem Statement

Before we proceed to the formal problem definition, let us state our assumptions:

1. Queries are long-running: They are submitted to the system and run until the user decides to interrupt query execution. In our work, we assume that the (collaborative) signal processing algorithms that produce sensor data are implemented and deployed separately from the queries formulated by users to access these data.
2. We define a query as a tree of operators[2]. The leaves of the trees correspond to the *source* nodes, i.e., the nodes that return data on behalf of the (collaborative) signal processing algorithms that produce *data streams*. The internal nodes of the trees are correlation, filtering, aggregation, or duplicate elimination *operators* that take as input one or several data streams and output another data stream. The root of the query tree is an operator that takes one data stream as input and forwards it; typically this root operator is placed on a gateway node that we denote *sink* in the rest of the paper. Note that we assume that there is one fixed sink per query. Relaxing this assumption is an interesting topic for future work.

Let us now proceed to the formal problem definition. We aim at placing operators on sensor nodes in order to minimize data transfer in the network. We consider a sensor network to be a directed graph where vertices represent sensor nodes and where edges represent communication links, and a query as a tree of operators. We define:

1. An oriented sensor network graph (SNG) as
   a) ζ - a set of sensor nodes. In the rest, p and q are elements of ζ.
   b) π - a set of communication links connecting the nodes in ζ. We denote (p, q), the link between nodes p and q, an element of π.
   c) w_{pq} - a weight is a positive integer associated with the link (p, q) of π
2. An oriented query tree (QT) as
   a) η - a set of operators. In the rest, operators i and j are elements of η.
   b) λ - a set of communication dependencies connecting the operators. We denote (i, j), the link between operators i and j, an element of λ. We denote i as the child and j as the parent in this communication link. Because QT is an oriented query tree, each operator has zero, one or more children and at most one parent.
   c) d_{ij} - a weight associated with the link (i, j) of λ

[2] Possibly such a tree of operators is generated from a declarative query language [2, 16]

w_{pq} denotes the unit cost of communicating data through the link (p, q). This cost might for example vary with the battery power of the sensor node. We define the cost of a path $P = \{(p, x), \cdots, (y, q)\}$ between node p and q as $C_p(P) = \sum_{e \in P} w_e$ – note that x and y are nodes in ζ. We denote the cheapest path between p and q by $P_{min}(p, q)$.

d_{ij} denotes the rate at which data is transmitted between operator i and j. Because queries are long-running, the rate at which an operator produces data might vary in time (e.g., more detections are produced or a correlation operator produces more data as its input become more correlated).

The *transfer cost* of sending data between operator i on node p and operator q on node j may be denoted by a function $S_{pq}(d_{ij})$ defined by $S_{pq}(d_{ij}) = C_p(P_{min}(p, q)) \cdot d_{ij}$, i.e., the transfer cost is a function of the path cost and the amount of data sent through the path.

A placement of a query tree onto a sensor network graph may be expressed as a mapping, i.e., a set $M = \{(i, p), ...\}$ where every operator $i \in \eta$ is assigned to a node $p \in \zeta$.

The placement problem can now be stated as the assignment of operators onto nodes that minimizes the following global cost:

$$\sum_{(i,j) \in \lambda} x_{ip} x_{jq} S_{pq}(d_{ij}) \tag{1}$$

subject to

$$\sum_{p \in \zeta} x_{ip} = 1, \forall i \in \eta, \forall p \in \pi : x_{ip} \in \{0, 1\} \tag{2}$$

where $x_{ip} = 1$, if operator i is assigned to node p, and $x_{ip} = 0$, otherwise. Equation (2) ensures that each operator is assigned to exactly one node of the network.

This problem of operator placement is an instance of the *task assignment problem*. The task assignment problem considers the problem of assigning a set of tasks onto a network of processors. The general task assignment problem is known to be NP-complete. Bokhari [1] has devised an $O(mn^2)$ algorithm for the case where the set of tasks is tree-structured, which is the case of our query tree (m denotes the number of taks and n the number of processors in the network). This algorithm is centralized; as a consequence it cannot be used in a sensor network.

3 Placement Strategy

We aim at defining a decentralized and adaptive algorithm for the placement of a query tree onto a sensor network. More precisely, our objective could be stated as follows: *Given an initial arbitrary placement of operators, our goal is to define a decentralized algorithm that progressively refine the placement of operators towards an optimal placement.*

It can be shown that the optimal placement of a query tree is composed of local optimal placements for each operator in the query tree. By *local optimal placement* we mean an assignment of operator i on node p that minimizes the amount of data that i receives from its children and transmits to its parent[3]. We omit the proof of this result because of lack of space.

This notion of local optimal placement constitutes an objective for the placement of individual operators. Now the question is: How can a decentralized algorithm move individual operators to their local optimal placement? Before we detail our algorithm in the next section, we give here the intuition behind our placement strategy.

If we disregard the limitations imposed by the sensor network topology in terms of operator placement and shortest paths, we may imagine that operators could be placed anywhere in a euclidean space. Data could be transferred along straight lines between operators. Transfer cost would be proportional to the distance between operators (multiplied by the transfer rate).

We could then view the transfer cost between two operators as a force pulling the operator towards one another. From the laws of physics we know that the equilibrium is the center of gravity of n particles. The net force determining the direction of the movement is the sum of the individual force vectors acting on the body. As the body moves in the direction dictated by the net force it reaches the optimal position through the shortest path (a straight line). The net force (and cost) will decrease monotonically along this path. In the equilibrium the net force acting on the body is zero and the cost is minimal. This equilibrium constitutes the local optimal operator placement. This analogy with forces has previously been used by Heiss and Schmitz [8] to develop a decentralized algorithm that achieves dynamic load balancing in a multicomputer system.

To understand the usefulness of these observations we now restrict operator positioning and possible paths between operators to those of a Manhattan graph (Figure 4(a)). Such a graph is a simplified but useful idealization of a wireless ad-hoc sensor network and is often used in analytic models (e.g. [15]). Possible operator positions are confined to the vertices of the graph. Assuming equal weights on links and equal data rates, the cheapest paths between operators corresponds to the path with the minimal Manhattan distance.

If an operator is not in the optimal position, there will exist a cheapest path between the operator and its local optimal position along which operator placement becomes progressively cheaper. An operator can reach its local optimal placement by walking this cheapest path. In a Manhattan graph, an operator initially placed on an arbitrary node will thus progressively reach its local optimal placement by greedily moving to the neighbor with lowest estimated cost. We call this placement strategy *neighbor exploration*.

We use the neighbor exploration as the placement strategy for the decentralized algorithm that we present in the next section. Note that in a sensor network with an arbitrary graph topology, the neighbor exploration policy might reach

[3] Recall that we denote i as the child and j as the parent in any communication link between operators $(i, j) \in \lambda$

local minima different from a global optimum. We present simulation results in Section 5 that show that our algorithm performs well in sensor networks with various graph topologies despite the potential pitfall of local minima.

4 Adaptative and Decentralized Operator Placement

Let us assume that an operator assignment has been defined for a given operator. We denote the node on which the operator is placed and executed, the *active* node. The sink and the sources constitute special active nodes. They run operators that respectively consume and produce data, they are involved in the algorithm and their placement is fixed.

As we have seen in the introduction, any assignment may become suboptimal. In order to adapt to changing conditions (e.g., data rate, data correlation), our decentralized algorithm implements the neighbor exploration strategy: It (i) evaluates the cost incurred by the execution of the operator at the active node, (ii) estimates the cost for alternative assignments of the operator, (iii) compares the cost on the active node and on alternative nodes, and (iv) transfers the operator to the node with lowest cost that thus becomes the new active node.

4.1 Decentralized Cost Computation

We define the cost of an operator j assigned to node q as:

$$C_o(j,q) = \sum_{(i,j)\in\lambda} (C_o(i,p) + S_{pq}(d_{ij})) \tag{3}$$

where variables and functions are as defined in Section 2.

Since equation (3) is recursive, the cost of an operator includes the accumulated cost of all operators delivering data for the operator. The cost of the query tree – defined in equation (1) - is the cost of the root operator.

A node needs the cost of the child operators in order to evaluate the cost of an operator. As a consequence child operators send cost messages to their parent operator along the data path. The term $S_{pq}(d_{ij})$ represents the cost of sending data between the child and parent operator. This cost may be accumulated while the cost message travels the path between node p and q if we assume (i) that the cost w_{xy} of sending data to a neighbor y is available at every node x and (ii) that we include the data rate d_{ij} with each cost message. The following equation states that the cost may be calculated by summing the product of the data rate with the link cost of each hop in the path[4]:

$$S_{pq}(d_{ij}) = C_p(P_{min}(p,q)) \cdot d_{ij} = \left(\sum_{e\in P_{min}} w_e \right) \cdot d_{ij} = \sum_{e\in P_{min}} (w_e \cdot d_{ij}) \tag{4}$$

[4] Note that though equation (3) refers to node mappings for technical reasons, information about the actual mapping on the child operator is not necessary for the parent to perform the cost calculation.

As a matter of fact the accumulated cost may simply be added to the cost of the child operator as the cost message travels to the parent operator, since the parent operator is only interested in the total sum. Cost messages flowing from each child to parent operator contain the accumulated cost and the data rate, e.g., the message sent by node i on node p to operator j on node q is: $(C_o(i,p), d_{ij})$. Any operator in the query tree will have information about the cost attributed by each of its sub-trees and will be able to calculate and forward its own cost.

4.2 Exploration

Exact data rates are necessary to calculate cost. The data rate of a given operator does not depend on its placement (but on the rate of the input data streams and on their contents). As a consequence the data rate calculated by operators on active nodes may be used for calculating costs on alternative nodes[5]

The idea is simply to have the active operators communicate their data rates to alternative nodes so that cost can be computed. Since there is a set of alternative nodes associated to each active node that only serve to probe the solution space for better alternatives we shall term them *tentative nodes*. These tentative nodes execute *tentative operators* responsible for the cost calculation. Cost is computed in the same way as for the active node. Cost messages flow from children (both active and tentative) to parents (both active and tentative) operators, while data flows from active children to active parent operators.

To explore the space of tentative nodes, we need to define:

1. An *exploration policy* for choosing what nodes should be elected tentative nodes. The sheer complexity of the solution space prohibits considering more than a tiny fraction, so the policy must choose the tentative nodes based on heuristics that will increase the probability of including an optimal, or at least better assignment in the face of data rate variations. Following our neighbor exploration strategy, only immediate neighbors of an active node are considered as its associated tentative nodes.
2. We also need an *adaptation policy* for choosing a new active node among the tentative nodes explored. We only consider two possible actions: either to continue query execution using the active node or switching to a new active node. Following our neighbor exploration strategy, the adaptation policy simply greedily picks the cheaper tentative node as the new active position since this would be closer to the optimal position than more expensive neighbors.

The close proximity of tentative nodes means that the communication overhead incurred by the transmission of data rates between the active and tentative nodes will be minimal. No tentative node is more than one hop away so multihop path establishment will not be necessary. If the MAC layer supports message

[5] Alternative methods would consist in executing several instances of the same operator, which would be costly if complete data streams were duplicated, and inaccurate if cost was estimated using a non-representative fraction of a data stream.

multicasting all tentative nodes may receive information from the active node using only one transmission.

The cost of an operator relative to its neighbors depends on the incoming as well as outgoing transfer cost. Since the outgoing transfer cost is not available until the transfer has actually been made through a path between the operator and its parent, it is natural to have the parent operator make decisions on active child operator assignments, i.e. the adaptation policy is executed by the active parent nodes.

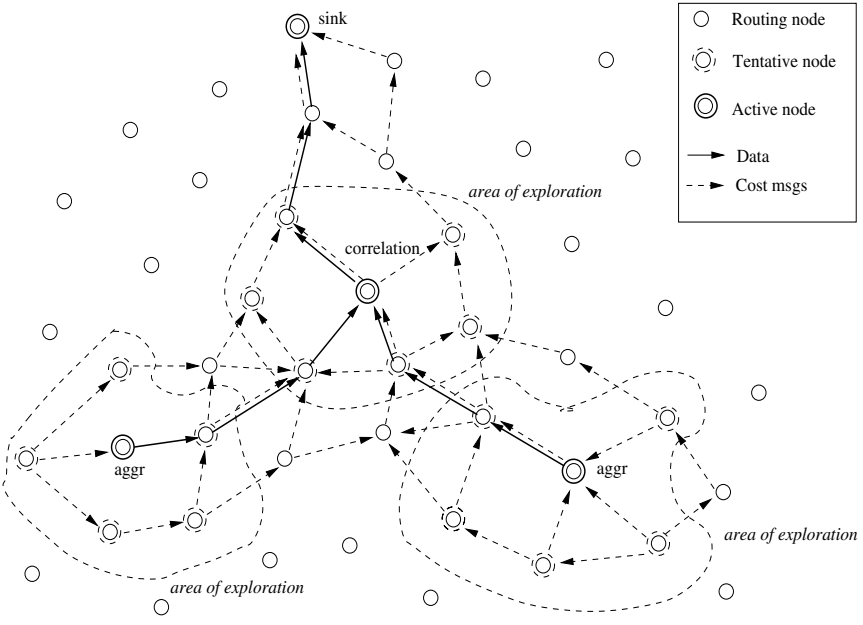


Fig. 2. Tentative and active Nodes

Figure 2 illustrates the flow of cost messages and data between active and tentative operators in the context of a complex query tree composed of an operator correlating the output of two aggregation operators.

4.3 Node Switching

At any point in time, there is only one active instance of each operator. The cost information received by the active parent operator allows it to implement the adaptation policy. When a tentative child operator instance has a lower estimated cost than the active child, the active operator may initiate a node switch. This switch consists of recursive signaling down two subtrees. Active operators in the active subtree must be informed that data flow is to cease. The

cheaper tentative operator, on the other hand, must be informed that active dataflow is to start. The cheaper tentative operator will propagate this signal to its cheapest child instances. When the signal reaches the leaves, the leaf operators then begin to send data through the new active path. We shall term the two signal types *activation* and *deactivation* respectively.

There is one more issue to active plan switching: we need to support operator state transfer. Long-running operators on continuous data streams often need historic or accumulated information for their operation. Aggregation operators may keep a sliding window of values [13,16] and correlation operators usually store two sets of tuples that are probed and updated when new tuples arrive [2]. For the transition to be seamless such information must be transferred between the old and new active operators. Since we have already assumed that data rates can be communicated from the active operator to tentative operators, the same channel may be used for operator state transfer[6].

Operator state transfer could also be used to replicate the state of the active operator so that tentative nodes could take over in case the active node fails or runs out of energy. Designing an efficient fault tolerant placement of operators is a topic for future work.

4.4 Summary

Our algorithm, based on the exchange of cost messages and data between active and tentative operators, is adaptive and decentralized. It is adaptive because the placement of active operators is continuously refined depending on the estimated cost on their associated tentative nodes. It is decentralized because decisions are taken at the level of each operator. The information maintained on each node is local: active nodes merely maintain information about their children. Cost messages are transmitted in the network in addition to the data streams. The frequency at which cost messages are exchanged is a parameter of our algorithm (resulting in a trade-off between the responsivity of operator placement and the transmission overhead).

Note that our algorithm does not dictate how data should be routed between operators. We can thus use any routing protocol that relies on logical naming of nodes; we are currently implementing our algorithm on top of directed diffusion [6], using the filter mechanism to implement cost computation.

Figure 3 summarizes the exchanges between active and tentative operators (both parents and children) in our neighbor exploration policy:

– An active operator is defined as the instance of an operator which is actually executed; it receives input data streams, process them and generates an output data stream. An active operator is located on an active node.

[6] The design of efficient mobile operators is beyond the scope of this paper. Topics for future work include the design of operators requiring minimal internal states, and the design of efficient mechanisms supporting the marshalling/unmarshalling of the internal state and ensuring the continuity of execution while an operator is being moved from one node to another.

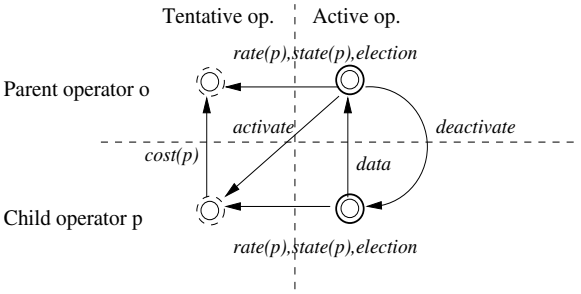


Fig. 3. Illustration of the communications between active and tentative operators

- A tentative operator is associated to an active operator in order to explore the cost of execution on alternative nodes (called tentative nodes). A tentative operator computes cost messages and transmits them to its parent. In order to compute cost, tentative operators receive data rate from their associated active node.
- Given the cost obtained from its children (both tentative and active), an active parent operator can decide to switch execution to a new active child operator. It then sends a deactivation message to the current active child operator and an activation message to the newly chosen active child operator. The current active child operator transfers its state to the new chosen active child operator.

5 Simulation Results

Our objective was to verify that the neighbor exploration strategy for operator placement is viable in sensor networks with various topologies. We focus on the placement of a single operator towards its local optimal placement, which is the basis of the neighbor exploration strategy.

The results we present in this section are essentially a proof-of-concept. We have now implemented our decentralized and adaptive algorithm on top of directed diffusion in order to measure the overhead incurred by our approach. This implementation allows us to experiment with the placement of complex query trees (with several operators) and with the adaption of the placement to changing conditions in the sensor network. Initial results confirm the good performances suggested by the simulations.

5.1 Simulation Framework

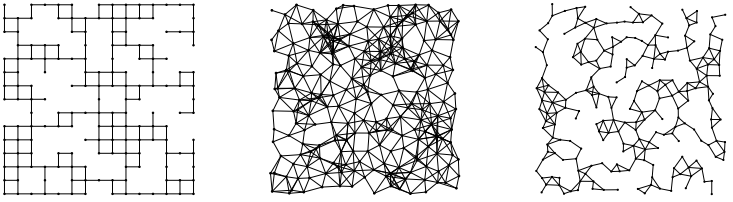
Topologies. Three types of basic network topologies were used in the simulations: (i) the maximal planar graph (MPG), (ii) the Manhattan graph (MG)

and (iii) controlled random graphs (CRG)[7]. These graphs represent both abstract and realistic sensor network topologies.

The topologies were gradually degraded in terms of connectivity. This was done in one of two ways: (i) for the MPG and MG by removing a percentage of the nodes at random and (ii) for CRG by increasing the area in which a number of nodes were deployed thereby reducing the number of reachable neighbors gradually (fig.4). The CRG was produced by spreading a fixed number of nodes across a square area at random. When placing a node we try to ensure that the distance to any other node is at least half of the reach of the node. This is to avoid unrealistically close nodes. If we don't succeed in a few attempts we place the node anywhere. The x and y dimensions of the area is calculated as:

$$\sqrt{n} \cdot R \cdot f$$

where R is the reach of node radios and f is a factor varied between 0.55 and 0.90. A number of topologies where the area is gradually increased this way produce quite realistic random networks with decreasing density (10 to <3.8 neighbors) (fig.4). At higher values of f connected networks becomes harder to generate and less realistic.



(a) Manhattan graph with 25% holes

(b) Generated random graph dense (f=0.60)

(c) Generated random graph less dense (f=0.90)

Fig. 4. Network Topologies

Query Plan. The query plan used for the simulations consists of two fixed sources, a correlation operator and a fixed sink at which the result is delivered. The two sources are equally productive and the correlation operator half as productive as the sources. We consider this to be slightly more challenging than having different rates of the source operators because the equal pull from the sources may get the operator stuck in a local optima, i.e., the operator could be stuck on one side of a hole in topologies with a source on either side of the hole.

[7] We term these graphs *controlled random* because we tried to avoid unrealistically close nodes during generation.

Simulator. For each type of topology and density, we generated 30 network instances with 225 nodes each. We ran 100 simulations on each network instance with random initial placements of the fixed sources and sink.

In order not to favor our scheme we have disregarded "easy" adaptation scenarios where the optimal assignment was less than 4 hops away from the initial operator assignment. In particular this does away with the trivial case where the operator is already in the optimal position.

For each run the cost of the optimal assignment, the assignment achieved through adaptation as well as the cost of data extraction was calculated. By data extraction we denote the case without in-network processing: each source sends the data it produces to the sink and data is processed outside the sensor network. These results were averaged for each topology type and density type and used for depicting the cost development of a topology as density decreases. This should provide a good statistical picture of the adaptation behaviour for a given topology and density.

The optimal assignment of the operator was found by performing an exhaustive search for cheapest position. Doing so was feasible because only one operator was at play. With complex query trees the mapping complexity would have required a more efficient method like that of Bokhari [1].

5.2 Results

Quality of Operator Placement. Figure 5(a) shows the cost of data extraction and the cost of optimal assignment together with the cost of the assignment achieved through neighbor exploration.

At high densities perfect or near-perfect adaptation is achieved. The dense random graphs are very likely to have Manhattan subgraphs explaining the initial coinciding graphs of optimal and adapted assignment costs.

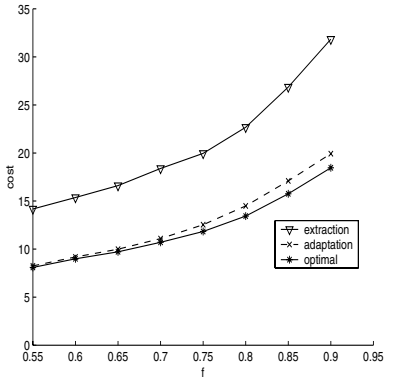
The somewhat surprising findings are that our simple scheme does very well even in the least dense topologies. In no case is the average cost deviation greater than 10% of the optimal cost.

The slightly increasing tendency of the curves is caused by longer average inter-operator shortest paths as the networks get less connected. I.e. the direct way to a neighbor operator becomes less direct. These longer paths result in higher data transfer costs.

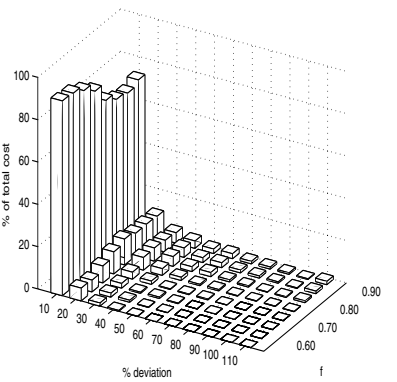
Since average measures says little about worst-cases, we also depict the constituents of the cost achieved through adaptation grouped by percentual deviation from the optimal cost on Figure 5(b)).

Even for the topologies of lowest density more than 70 % of the cost is attributed by adaptations deviating less than 10 % from the optimal assignment. We obtained equally good results with MG and MPG topologies but we omit the figures because of space constraints.

In our simulations the extraction cost was approximately twice the optimal cost due to the network size, the specific number of operators, their productivity and selectivity. We have been conservative with these parameters not to



(a) Extraction, adaptation and optimum



(b) Cost distribution

Fig. 5. Cost random graphs

favor our approach unreasonably. Still our results indicate that in-network query processing is very promising.

6 Related Work

The notion of in-network processing in a sensor network was first introduced in order to opportunistically eliminate duplicates in the context of directed diffusion [6]. Intanagonwiwat et al. [11] extended this work by constructing a routing tree where paths are shared as much as possible to increase the possibilities of eliminating duplicates. The potential benefits of in-network duplicate elimination have also been studied from a theoretical perspective [12]. By comparison, we consider queries that embed richer application-level data processing (correlations, filters, aggregates). Those query trees are fixed a priori, which imposes a strong constraint on the routing tree. Also the cost function that we consider for operator placement includes the rate at which data is transmitted across operators in addition to the length of the path between those operators.

Madden et al. [13] were the first to study in-network query processing. They focused on simple aggregation queries, whose execution can be distributed over an arbitrarily large set of operators. They defined both (i) aggregate operators adapted to motes with limited ressources running tinyOS, and (ii) a routing strategy that imposes a spanning tree onto the network: data is aggregated at every internal node in the routing tree. Note that they assume that queries are submitted in a declarative, SQL-like form. The placement of the query tree is constrained by the characteristics of the routing tree. We have taken the alternative approach where the routing tree is constrained by the query tree and

where data is processed only at a few nodes. Our approach is particularly relevant when the query contains *holistic aggregates* such as correlations or median [13] or materialization operators such as storage points [9].

Recently, Yao and Gehrke [16] have devised a general framework for in-network query processing. Each query is decomposed into *flow blocks* determining a set of sensor nodes that elect a leader on which a query fragment is executed. They applied this strategy to queries complex aggregates (with group by clauses) as well as joins (similar to our correlation operator). Our approaches are very much complementary in the sense that they defined a general framework for the optimization of declarative queries into query trees that could be a way to generate our query tree. Also, our algorithm could be seen as an election protocol, particularly well-suited for the adaptive placement of flow block leaders.

7 Conclusion

The problem of operator placement is crucial for in-network query processing. We showed that it was a variant of the task assignment problem and we described an adaptive and decentralized algorithm based on the *neighbor exploration* strategy: the placement of operators is progressively refined from neighbor node to neighbor node until a local optimal placement is reached. Simulation results stress the potential benefits of in-network query processing. They also show that neighbor exploration can achieve near optimal placement of a single operator with various graph topologies, despite the risks of local minima.

Future work includes a complete performance study of our algorithm. We have implemented it on top of directed diffusion and we have started to run experiments using NS. These experiments include a measure of the communication overhead introduced by our algorithm, as well as measures of the quality of adaptation with changing network conditions for single operators as well as complex query trees.

Other topics for future work are the design of query operators that can be moved from node to node with minimal overhead as well as the design of fault tolerant operator placement algorithms.

Acknowledgements. We would like to thank Mads Dydensborg, Stefan Røpke, Jacob Simonsen and Christian Stefansen for their help debugging this paper as well as Pawel Winter and Sam Madden for interesting discussions.

References

1. S.H. Bokhari: A shortest Tree Algorithm for optimal Assignments Across Space and Time in a Distributed Processor System. *IEEE Transactions on Software Engineering, Vol. se-7 no. 6* , November 1981.
2. Ph. Bonnet, J. Gehrke and P. Seshadri. Towards sensor database systems. *2nd International Conference on Mobile Data Management*, Hong Kong, Jan. 2001.

3. N.H. Cohen, A. Purakayastha, J. Turek, L. Wong, D. Yeh. Challenges in Flexible Aggregation of Pervasive Data. *IBM Research Division*, January 2001.
4. D. Estrin, R. Govindan, and J. Heidemann. Embedding the internet: Introduction. *Communications of the ACM*, 43(5), 2000.
5. D. Estrin, M. Srivastava, and A. Sayeed. Tutorial on Wireless Sensor Networks *ACM Mobicom*, Sep. 2002.
6. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *SOSP*, 2001.
7. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient Communication Protocol for Wireless Microsenset Network *Hawaii International Conference on System Sciences*, 2000.
8. H-U. Heiss, and M. Schmitz. Decentralized Dynamic Load Balancing: The Particles Approach. *Information Sciences 84(1&2*. 1995.
9. J. Hellerstein, W. Hong, S. Madden, and K. Stanek Beyond Average: Towards Sophisticated Sensing with Queries *IPSN*, 2003.
10. C. Intanagonwiwat, R. Govindan, D. Estrin: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks, Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (Mobicom 2000), August 2000, Boston, MA.
11. C. Intanagonwiwat, R. Govindan, D. Estrin, and J. Heiderman. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. *DEBS'02*
12. B. Krishnamachari, D. Estrin, S. Wicker. The Impact of Data Aggregation in Wireless Sensor Networks *Distributed Event Based Systems*, 2002.
13. S. Madden, M.J. Franklin, and J. Hellerstein. TAG: a Tiny AGregation Service for Ad-Hoc Sensor Networks. *OSDI*, 2002.
14. G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.
15. S.D. Servetto, G. Barrenechea: Constrained Random Walks on Random Graphs: Routing Algorithms for Large Scale Wireless Sensor Networks, First ACM International Workshop on Wireless Sensor Networks & Applications, September 2002, Atlanta, Georgia.
16. Y. Yao and J. Gehrke. Query Processing in Sensor Networks *First Biennal Conference on Innovative Data Systems Research. CIDR 2003.*.
17. F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, March 2002.