Operator Placement for In-Network Stream Query Processing

Utkarsh Srivastava
Stanford University
usriv@cs.stanford.edu

Kamesh Munagala
Duke University
kamesh@cs.duke.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

Abstract

In sensor networks, data acquisition frequently takes place at low-capability devices. The acquired data is then transmitted through a hierarchy of nodes having progressively increasing network bandwidth and computational power. We consider the problem of executing queries over these data streams, posed at the root of the hierarchy. To minimize data transmission, it is desirable to perform "in-network" query processing: do some part of the work at intermediate nodes as the data travels to the root. Most previous work on in-network query processing has focused on aggregation and inexpensive filters. In this paper, we address in-network processing for queries involving possibly expensive conjunctive filters, and joins. We consider the problem of placing operators along the nodes of the hierarchy so that the overall cost of computation and data transmission is minimized. We show that the problem is tractable, give an optimal algorithm, and demonstrate that a simpler greedy operator placement algorithm can fail to find the optimal solution. Finally we define a number of interesting variations of the basic operator placement problem and demonstrate their hardness.

1 Introduction

We consider query processing in environments where data is collected at "edge devices" with limited capabilities, such as sensors. Collected data is transmitted through a hierarchy of network nodes and links with progressively increasing computational power and network bandwidth, as shown in Figure 1. The "high fan-in" environment addressed by the Berkeley *HiFi* project [10] is one example, but other scenarios that involve data acquisition and subsequent processing, e.g., network monitoring [6], may exhibit similar general characteristics.

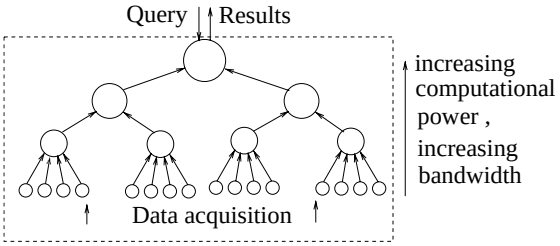

Figure 1: Sensor Data Processing

Typically, queries are posed and results collected at the root of the hierarchy. One simple approach is to transmit all data acquired at the sensors through the hierarchy to the root, then perform all query processing at the root. However, if queries produce significantly less data than they consume—because of filtering, aggregation, or low-selectivity joins—then this approach may pose considerable unnecessary burden on network bandwidth. *In-network query processing* pushes some or all of the query execution task to nodes lower in the hierarchy, in order to reduce overall network costs [15]. In general, finding the best point in the hierarchy at which to perform specific processing is a difficult problem: Placing work lower in the hierarchy reduces transmission costs but imposes more burden on lower-capability devices. The goal is to properly balance these opposing effects to minimize overall cost.

Previous work on in-network query processing has focused on queries in which data reduction occurs because of aggregation operators [15], or through inexpensive filtering that is not a burden on edge devices (so all filters are pushed to the leaves). We consider queries that may involve expensive predicates, such as text, image, or video filtering, or lookups to remote sources. For these operators it may not be best (or even possible) to filter at low-capability sen-

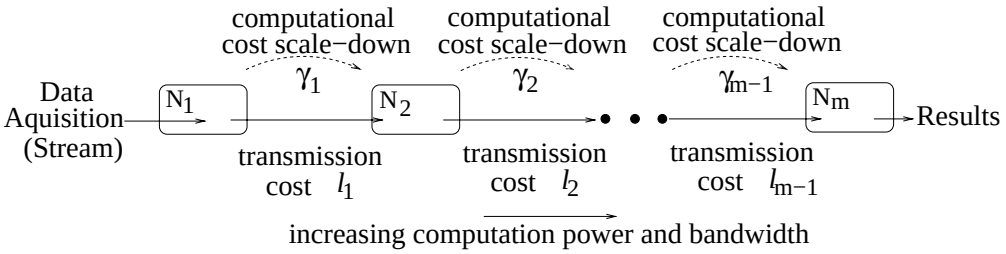

Figure 2: Basic Scenario and Cost Model

sors. Our objective is to place each filter operator at the "best" node in the hierarchy for that filter, based on its selectivity and cost, and considering the computational capabilities of the nodes up the hierarchy against the burden of bringing the data to each node.

Suppose we have an m-level hierarchy and n filters in our query. For one sensor's input stream there are m^n possible filter placements for processing the data as it travels to the root. We show that nevertheless the operator placement problem has a polynomial-time optimal solution. We provide an optimal algorithm, and show that a simpler greedy algorithm can fail to find the optimal solution.

A key idea in our work is to model network links as filters. Then we can address our overall problem as one of filter ordering on a single node, but with precedence constraints for those filters that are modeling links. We start by considering *uncorrelated* filters, i.e., filters whose selectivity is independent of the other filters, and then extend our algorithm to correlated filters. In both cases, we show how the precedence constraints can be dealt with so that known results on filter ordering [13, 17] can be reused. After addressing queries with filters alone, we extend our algorithm to include multiway joins, showing how to decide where a join operator should be placed optimally with respect to the query's filter operators.

The overall contributions of this paper are:

- We define the problem of operator placement for in-network processing of queries with expensive filters (Section 2).
- We describe a greedy algorithm that can fail to find the globally optimal solution to the operator placement problem, then present a polynomial-time optimal algorithm for uncorrelated filters. We extend our algorithm to provide the best possible approximation for correlated filters (Section 3).
- We extend our algorithm to include operator placement for a multiway stream join together with filters (Section 4).
- We identify several variations on the problem and in some cases show their hardness (Section 5). We consider nodes with resource constraints, load balancing across nodes, and a more complex cost model for how filter costs may vary across different nodes.

Related work and conclusions are presented in Sections 6 and 7.

2 Preliminaries

We begin by considering data acquired by only one of the leaf nodes of Figure 1 and focus on in-network query processing over this data. As this data is transmitted up the hierarchy, the basic network topology we need to consider (shown in Figure 2) consists of a linear chain of nodes $N_1, N_2, \ldots, N_m$, where m is the number of levels in the hierarchy. In relation to Figure 1, the leftmost node N_1 corresponds to the point of acquisition, while the rightmost node N_m corresponds to the root of the hierarchy. Each node N_j transmits only to node N_{j+1}. We consider the linear hierarchy merely for ease of presentation; in Section 3.4 we show how our algorithms extend in a straightforward manner to general tree hierarchies.

Let stream S denote the data acquired by node N_1. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ be a set of n filters. We first consider in-network processing for the following basic query posed at the root node N_m.

$$\texttt{SELECT} * \texttt{FROM S WHERE } \texttt{F}_1 \wedge \texttt{F}_2 \wedge \ldots \wedge \texttt{F}_\texttt{n} \quad (1)$$

In Section 4 we extend our algorithms to deal with queries that involve a multiway join of streams in addition to conjunctive filters. In this paper we do not

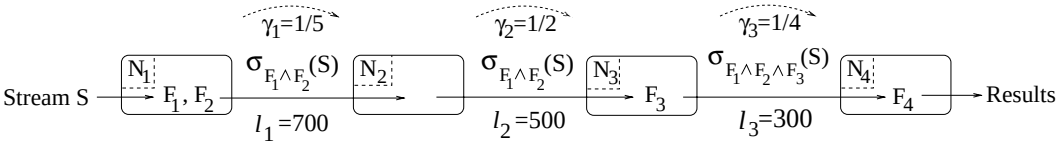

Figure 3: Running Example

consider multiple queries together: The possibility of shared computation among multiple queries yields an even more complex operator placement problem that we leave as future work.

An *in-network query plan* for the query in (1) is simply a mapping of each filter in $\mathcal{F}$ to exactly one node. Figure 3 shows a sample in-network query plan for executing a query with $n = 4$ filters on $m = 4$ nodes. Figure 3 also shows the data that is transmitted along each network link. Each link transmits only those tuples that have passed all filters executed so far. The cost of an in-network query plan consists of two parts: the cost of executing the filters on the various nodes, together with the cost of transmitting the tuples over the network links. The exact model used to evaluate the cost of an in-network query plan is explained in the next section.

2.1 Cost Model

The cost of an in-network query plan is calculated using the following three quantities:

1. **Selectivity of filters**: Associated with each filter F is a selectivity $s(F)$ that is defined as the fraction of the tuples in stream S that are expected to satisfy F. We assume for now that the filters are independent, i.e., selectivity of a filter remains the same irrespective of which filters have been applied earlier. Correlated filters are dealt with in Section 3.3.

2. **Cost of filters**: Each filter F has a per-tuple cost $c(F, i)$ of execution on node N_i. To model the fact that the nodes in the hierarchy have increasing computational power, we assume that the cost of any filter scales down by a factor $\gamma_i < 1$ on moving from node N_i to N_{i+1} (see Figure 2). That is, $c(F, i+1) = \gamma_j c(F, i)$. Note that even though we are supposing scale-down, a decrease in computational power on moving from node N_i to N_{i+1} is captured by $\gamma_i > 1$ and can be incorporated into our approach directly.

3. **Cost of network transmission**: The cost of transmitting a tuple on the network link from node N_i to N_{i+1} is l_i (see Figure 2). We assume that l_i includes an appropriate multiplicative factor to convert transmission cost into a quantity that can be treated at par with computational cost.

Consider an in-network query plan $\mathcal{P}$ for the query in (1). Let $\mathcal{P}(F)$ denote the index number of the node at which filter F is executed under plan $\mathcal{P}$. Let $\mathcal{F}_i$ be the set of filters executed at node N_i, i.e., $\mathcal{F}_i = \{F \mid \mathcal{P}(F) = i\}$. We assume that at each node N_i, the set of filters $\mathcal{F}_i$ are executed in the optimal sequence given by the following theorem [13].

Theorem 2.1. *The optimal sequence to execute a set of independent filters on a single node is in increasing order of* rank, *where rank of a filter F is given by* $\mathit{rank}(F) = \frac{\mathit{cost}(F)}{1-\mathit{selectivity}(F)}$. □

Consider a sequence of n' filters $\mathcal{F}' = F'_1, \ldots, F'_{n'}$. Let $c(\mathcal{F}', i)$ denote the cost per tuple of executing this sequence at node N_i. It is given by:

$$c(\mathcal{F}', i) = \sum_{j=1}^{n'} \left(c(F'_j, i) \prod_{k=1}^{j-1} s(F'_k) \right) \qquad (2)$$

Let $r(\mathcal{F}_i)$ denote the sequence of filters in $\mathcal{F}_i$ in rank order.[1] Without loss of generality assume that the data in stream S is acquired at the rate of one tuple per some unit time. Then the cost per unit time of the in-network plan $\mathcal{P}$ is given by (assume $l_0 = 0$):

$$c(\mathcal{P}) = \sum_{i=1}^{m} \prod_{F \mid \mathcal{P}(F) < i} s(F) \Big(l_{i-1} + c\big(r(\mathcal{F}_i), i\big) \Big) \qquad (3)$$

Example 2.2. *Consider the in-network query plan shown in Figure 3. Let the selectivity of each filter be* $1/2$*, and let the costs at node* N_1 *of the filters be:*

[1] Since the filters have different costs at different nodes, the actual rank of a filter is node-dependent. However, since the cost of each filter scales by the same factor going from one node to the next, the rank order of filters remains the same at every node. In Section 5.2 we discuss a more general model in which each filter's cost may scale differently across nodes.

F	F_1	F_2	F_3	F_4
$c(F,1)$	200	400	1300	2500

The cost scaling factors and the transmission costs are as shown in Figure 3. Assume stream tuples are acquired at N_1 at unit rate.

Using equation (2)*, the execution cost of the sequence F_1, F_2 of filters at node N_1 is $200 + \frac{1}{2} \cdot 400 = 400$. Since two filters each with selectivity $1/2$ have been applied, the rate of data transmitted from N_1 to N_2 and from N_2 to N_3 is $1/4$ of the unit rate each. Thus the total transmission cost up to node N_3 is $\frac{1}{4}(700 + 500) = 300$. The per-tuple execution cost of F_3 at N_3 is $c(F_3, 3) = \gamma_1\gamma_2 c(F_3, 1) = 130$. Since the rate into N_3 is $1/4$, the execution cost of F_3 is $\frac{1}{4} \cdot 130 = 32.5$. Similarly the transmission cost from N_3 to N_4 and the execution cost of F_4 are calculated to be 37.5 and 7.8 respectively. Thus the total cost is $c(\mathcal{P}) = 400 + 300 + 32.5 + 37.5 + 7.8 = 777.8$.* □

2.2 Problem Statement

Since each of the n filters can be placed at any of the m nodes, there are m^n possible in-network query plans. The problem of operator placement for in-network query processing is to efficiently choose the least-cost plan among the exponential number of alternatives.

Definition 2.3 (Operator Placement Problem). *For each filter $F \in \mathcal{F}$, choose $\mathcal{P}(F) \in \{1, \ldots, m\}$ such that $c(\mathcal{P})$ given by* (3) *is minimized.* □

3 Filter Placement

In this section, we consider solutions to the operator placement problem given by Definition 2.3. We first assume independent filters and specify a local greedy operator placement algorithm (Section 3.1). We show that this algorithm does not always find the globally optimal solution. We then provide an optimal operator placement algorithm (Section 3.2), and extend this algorithm for correlated filters (Section 3.3) and tree hierarchies (Section 3.4).

3.1 Greedy Algorithm

For an in-network query plan $\mathcal{P}$, let $c(\mathcal{P}, i)$ denote the part of the total cost $c(\mathcal{P})$ that is incurred at node N_i. This cost includes not only the execution of filters at node N_i, but also the transmission of the filtered tuple stream from node N_i to N_{i+1}. $c(\mathcal{P}) = \sum_{i=1}^{m} c(\mathcal{P}, i)$, and notice that $c(\mathcal{P}, i)$ depends only on $\mathcal{F}_1, \ldots, \mathcal{F}_i$ and not on $\mathcal{F}_{i+1}, \ldots, \mathcal{F}_m$.

A simple but reasonable way to approach the operator placement problem is the following greedy algorithm. Start with node N_1 and choose a set of filters $\mathcal{F}_1$ so that $c(\mathcal{P}, 1)$ is minimized (explained in the next paragraph). Then apply the approach recursively with nodes $\{N_2, \ldots, N_m\}$ and the set of filters $\mathcal{F} - \mathcal{F}_1$. Our global objective is to minimize $\sum_{i=1}^{m} c(\mathcal{P}, i)$; the greedy algorithm minimizes each $c(\mathcal{P}, i)$ individually in increasing order of i. In other words, the greedy algorithm decides which filters to apply by balancing filtering cost against the cost of transmitting unfiltered data to the next node, but it does not take into account how much cheaper it would be to filter the data further up the hierarchy.

For minimizing $c(\mathcal{P}, 1)$ in the base case of the recursion, we introduce a key idea behind all our algorithms: modeling network links as filters. Logically, we construct a filter corresponding to each network link, such that transmitting a tuple over the link is equivalent in terms of cost to executing the constructed filter over the tuple. For cost evaluation, the entire in-network query plan can then be treated as executing a sequence of filters on a single node, enabling us to leverage previous work on filter ordering [3, 13, 17].

To minimize $c(\mathcal{P}, 1)$, model the network link from node N_1 to N_2 as a filter F_1^l with $s(F_1^l) = 0$ and $c(F_1^l, 1) = l_1$. We now show that $c(\mathcal{P}, 1)$ can be written as the cost of executing the filters in $\mathcal{F}_1$ followed by the filter F_1^l at node N_1.

Lemma 3.1. *Construct F_1^l with $s(F_1^l) = 0$, $c(F_1^l, 1) = l_1$. Then $c(\mathcal{P}, 1) = c\big(r(\mathcal{F}_1) \bullet F_1^l, 1\big)$ where $\bullet$ denotes concatenation of sequences.*

Proof. From (3),

$$\begin{aligned} c(\mathcal{P}, 1) &= c\big(r(\mathcal{F}_1), 1\big) + \prod_{F \in \mathcal{F} | \mathcal{P}(F) < 2} s(F) l_1 \\ &= c\big(r(\mathcal{F}_1), 1\big) + \prod_{F \in \mathcal{F}_1} s(F) c(F_1^l, 1) \\ &= c\big(r(\mathcal{F}_1) \bullet F_1^l, 1\big) \end{aligned}$$

□

We then order F_1^l and the filters in $\mathcal{F}$ based on rank (recall Theorem 2.1) and choose as $\mathcal{F}_1$ all the filters

that occur before F_1^l in rank order. Note that since rank$(F_1^l) = l_1$, effectively we simply choose as $\mathcal{F}_1$ all filters that have rank $< l_1$.

Theorem 3.2. *To minimize $c(\mathcal{P}, 1)$, $\mathcal{F}_1 = \{F \mid F$ occurs before F_1^l in $r(\mathcal{F} \cup \{F_1^l\})\}$ where all ranks are calculated at node N_1.*

Proof. Suppose $\mathcal{F}_1$ is chosen according to the theorem statement. By Lemma 3.1, $c(\mathcal{P}, 1) = c\big(r(\mathcal{F}_1) \bullet F_1^l, 1\big)$. Since $s(F_1^l) = 0$, we can append any number of filters after F_1^l without changing the cost of executing the sequence. Thus we can write

$$c(\mathcal{P}, 1) = c\big(r(\mathcal{F}_1) \bullet F_1^l \bullet r(\mathcal{F} - \mathcal{F}_1), 1\big) \qquad (4)$$

Now suppose for contradiction that there is a different set of filters $\mathcal{F}_1'$ to be executed at node N_1 and a corresponding in-network query plan $\mathcal{P}'$ such that $c(\mathcal{P}', 1) < c(\mathcal{P}, 1)$. Similar to (4), we can write

$$c(\mathcal{P}', 1) = c\big(r(\mathcal{F}_1') \bullet F_1^l \bullet r(\mathcal{F} - \mathcal{F}_1'), 1\big) \qquad (5)$$

The right sides of (4) and (5) give the execution cost of the same set of filters $\mathcal{F} \cup \{F_1^l\}$ but in different sequences. By the choice of $\mathcal{F}_1$, the sequence in (4) is rank ordered, but that in (5) is not. By Theorem 2.1, $c(\mathcal{P}, 1) < c(\mathcal{P}', 1)$. Thus we get a contradiction. □

We illustrate the operation of the greedy algorithm by an example.

Example 3.3. *Consider operator placement using the greedy algorithm for Example 2.2. The ranks of $F_1, \ldots, F_4$ at N_1 are 400, 800, 2600, and 5000 respectively. The rank of F_1^l is $l_1 = 700$. Thus $\mathcal{F}_1$ is chosen as $\{F_1\}$. The ranks of $F_2, \ldots, F_4$ at N_2 are obtained by scaling down the ranks at N_1 by γ_1, so they are $160, 520$, and 1000. Only rank$(F_2) < l_2$, thus $\mathcal{F}_2 = \{F_2\}$. Continuing in this fashion, we obtain $\mathcal{F}_3 = \{F_3\}$ and $\mathcal{F}_4 = \{F_4\}$. For this plan, we find $c(\mathcal{P}) = 792.8$ by* (3). □

The greedy algorithm makes very local decisions. Thus it is not surprising that the greedy algorithm does not always produce the globally optimal solution. For instance, $c(\mathcal{P}) = 792.8$ in Example 3.3 is greater than $c(\mathcal{P}) = 777.8$ in Example 2.2.

3.2 Optimal Algorithm

In the greedy algorithm of Section 3.1, network links are modeled as filters with selectivity 0. This approach enables us to capture the transmission cost of the link, but the remainder of the tuple processing cost (at nodes further up in the hierarchy) is not captured. Thus we can only get an expression for $c(\mathcal{P}, 1)$ in terms of the execution cost of a sequence of filters (Lemma 3.1), but not an expression for the entire $c(\mathcal{P})$. The optimal algorithm we present relies on obtaining an analogous expression for $c(\mathcal{P})$.

Assume $\gamma_i \leq 1$ for each i ($\gamma_i > 1$ is handled in Section 3.2.2). Since $c(F, i+1) = \gamma_i c(F, i)$, transmitting data on the link from node N_i to N_{i+1} cuts down by a factor γ_i the per-tuple cost of any filter applied subsequently. In terms of cost per unit time, this cost scale-down is equivalent to the stream rate slowing down by a factor γ_i, but the filter costs themselves remaining unchanged. Hence the link from node N_i to N_{i+1} can be modeled as a filter F_i^l with $s(F_i^l) = \gamma_i$. Additionally, we set $c(F_i^l, 1) = l_i(\prod_{j=1}^{i-1} \gamma_j)^{-1}$. Intuitively, the per-tuple cost of traversing the link is l_i, even after the previous network links have been traversed. The term $(\prod_{j=1}^{i-1} \gamma_j)^{-1}$ merely compensates for the scale-down produced by filters $F_1^l, \ldots, F_{i-1}^l$. We can now write $c(\mathcal{P})$ in terms of the execution cost of a sequence of filters (assume all ranks are calculated at N_1).

Lemma 3.4. *For $i \in \{1, \ldots, m-1\}$ construct F_i^l with $s(F_i^l) = \gamma_i$, $c(F_i^l, 1) = l_i(\prod_{j=1}^{i-1} \gamma_j)^{-1}$. Then $c(\mathcal{P}) = c\big(r(\mathcal{F}_1) \bullet F_1^l \bullet r(\mathcal{F}_2) \bullet \ldots \bullet F_{m-1}^l \bullet r(\mathcal{F}_m), 1\big)$ where $\bullet$ denotes concatenation of sequences.*

Proof. The result follows by noting that $c(r(\mathcal{F}_i), i) = \prod_{j=1}^{i-1} \gamma_j \cdot c(r(\mathcal{F}_i), 1)$ and performing suitable manipulation of $c(\mathcal{P})$ (given by (3)) as in Lemma 3.1. The detailed proof is omitted. □

Suppose for now that the ranks of the sequence of filters $F_1^l, \ldots, F_{m-1}^l$ (modeling links) are in non-decreasing order. Then we have the following result analogous to Theorem 3.2. The proof is very similar to that of Theorem 3.2 and hence is omitted.

Lemma 3.5. *Suppose rank$(F_i^l) <$ rank(F_{i+1}^l) for each $i \in \{1, \ldots, m-2\}$. Denote by $\mathcal{F}'$ the filter sequence $F_0^l \bullet r(\mathcal{F} \cup \{F_1^l, \ldots, F_{m-1}^l\}) \bullet F_m^l$. Then $c(\mathcal{P})$ is minimized when:*

$$\mathcal{F}_i = \{F \mid F \text{ occurs between } F_{i-1}^l \text{ and } F_i^l \text{ in } \mathcal{F}'\} \quad \square$$

In general the ranks of $F_1^l, \ldots, F_{m-1}^l$ may not be in non-decreasing order. To deal with such cases, we introduce the concept of "short-circuiting".

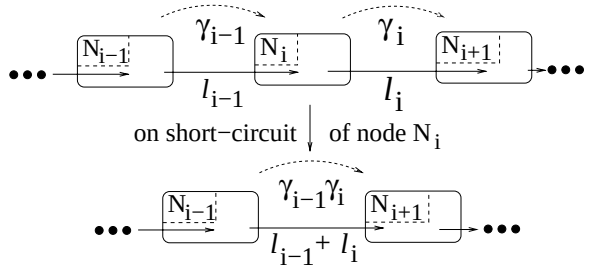

Figure 4: Short-Circuiting

3.2.1 Short-Circuiting

Suppose rank(F_{i-1}^l) > rank(F_i^l) for some i. We show that in the optimal in-network query plan in this scenario, no filter is executed at node N_i.

Lemma 3.6. *If rank(F_{i-1}^l) > rank(F_i^l) for some $i \in \{2, \ldots, m-1\}$, then in the optimal plan $\mathcal{F}_i = \emptyset$.*

Proof. Suppose rank(F_{i-1}^l) > rank(F_i^l) and in the optimal in-network query plan $\mathcal{P}$, $\mathcal{F}_i \neq \emptyset$. Consider the alternate query plans $\mathcal{P}'$ and $\mathcal{P}''$ where the filters in $\mathcal{F}_i$ have been moved to node N_{i-1} and N_i respectively. We have

$$c(\mathcal{P}) = a_1\Big(l_{i-1} + c(r(\mathcal{F}_i), i) + a_2 l_i\Big) + a_3$$

where $a_1 = \prod_{F|\mathcal{P}(F)<i} s(F)$, $a_2 = \prod_{F|F\in\mathcal{F}_i} s(F)$, and a_3 denotes the sum of the other terms in $c(\mathcal{P})$ from (3). Similarly:

$$\begin{aligned} c(\mathcal{P}') &= a_1\Big(c(r(\mathcal{F}_i), i)\gamma_{i-1}^{-1} + a_2(l_{i-1} + l_i)\Big) + a_3 \\ c(\mathcal{P}'') &= a_1\Big(l_{i-1} + l_i + \gamma_i c(r(\mathcal{F}_i), i)\Big) + a_3 \end{aligned}$$

Since $\mathcal{P}$ is optimal, we must have $c(\mathcal{P}) < c(\mathcal{P}')$ and $c(\mathcal{P}) < c(\mathcal{P}'')$. Substituting for $c(\mathcal{P})$, $c(\mathcal{P}')$, and $c(\mathcal{P}'')$ and simplifying, we get:

$$\frac{l_{i-1}\gamma_{i-1}}{1-\gamma_{i-1}} < \frac{l_i}{1-\gamma_i} \tag{6}$$

(6) implies that rank(F_{i-1}^l) < rank(F_i^l) which is a contradiction. □

If $\mathcal{F}_i$ is guaranteed to be empty in the optimal query plan, we can modify the network topology by "short-circuiting" node N_i as shown in Figure 4. Logically, node N_i is removed, N_{i-1} is connected to node N_{i+1} by a link having cost $l_{i-1}+l_i$, and the cost scale-down factor from node N_{i-1} to N_{i+1} is set to $\gamma_{i-1}\gamma_i$. At each short-circuit the number of nodes m decreases by 1.

Algorithm ***OPT_FILTER***
1. while $(\exists i \mid \gamma_i > 1)$
2. short-circuit node N_{i+1}
3. while (true)
4. for $i = 1$ to $m-1$
5. $s(F_i^l) = \gamma_i$ and $c(F_i^l, 1) = l_i(\prod_{j=1}^{i-1}\gamma_j)^{-1}$
6. if $\big(\exists i \mid$ rank$(F_{i-1}^l) >$ rank$(F_i^l)\big)$
7. short-circuit node N_i
8. else break
9. $\mathcal{F}' = F_0^l \bullet r(\mathcal{F} \cup \{F_1^l, \ldots, F_{m-1}^l\}) \bullet F_m^l$
10. for $i = 1$ to m
11. $\mathcal{F}_i = \{F \mid F$ occurs between F_{i-1}^l and F_i^l in $\mathcal{F}'\}$

Figure 5: Optimal Operator Placement Algorithm

We can continue short-circuiting on the modified topology until there does not exist any i for which rank(F_{i-1}^l) > rank(F_i^l). At that point, Lemma 3.5 can be applied to yield the optimal solution.

3.2.2 Handling Cost Scaleup

So far we have assumed $\gamma_i \leq 1$ for each i. If $\gamma_i > 1$, it is easy to see that in the optimal solution $\mathcal{F}_{i+1} = \emptyset$, as follows. If any filters are executed at node N_{i+1} they can be moved to node N_i. The new plan will reduce the computational cost (since $c(F, i) < c(F, i+1)$) as well as the transmission cost (since more filters are applied earlier reducing the amount of data transmitted). Thus, just as in Section 3.2.1, if $\gamma_i > 1$, we can short-circuit node N_{i+1} (if $\gamma_{m-1} > 1$ we can simply delete node N_m). We can continue short-circuiting until $\gamma_i \leq 1$ for each i.

3.2.3 Summary and Example

A summary of the entire algorithm is given in Figure 5. Its running time is $O((m+n)\log(m+n))$ due to the sorting of filters in rank order in line 9.

Example 3.7. *Continue with Example 2.2. We first construct a filter for each network link (line 5):*

i	1	2	3
$c(F_i^l, 1)$	*700*	*2500*	*3000*
$s(F_i^l)$	1/5	1/2	1/4
rank(F_i^l)	875	5000	4000

We find that rank$(F_2^l) > rank(F_3^l)$. Thus, we can short-circuit N_3 (line 7). On short-circuiting, we

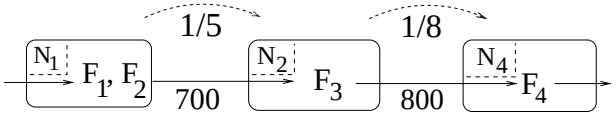

Figure 6: Optimal Plan for Example 2.2 (after short-circuiting N_3)

obtain a new link with transmission cost 800 *and scale-down factor* $1/8$ *(Figure 6). The filter corresponding to this link (denote it by* $F^l_{2,4}$*) has cost* 4000*, selectivity* $1/8$ *and hence rank* 4571.4*. Since rank*$(F^l_1) <$ *rank*$(F^l_{2,4})$*, no more short-circuiting is required. The ranks of* $F_1, \ldots, F_4$ *are* 400*,* 800*,* 2600*, and* 5000*. Thus the rank order of filters is* $F_1, F_2, F^l_1, F_3, F^l_{2,4}, F_4$ *(line 9).* $\mathcal{F}_1 = \{F_1, F_2\}$*,* $\mathcal{F}_2 = \{F_3\}$*, and* $\mathcal{F}_4 = \{F_4\}$ *(line 11). Since* N_3 *has been short-circuited,* $\mathcal{F}_3 = \emptyset$*. For this plan,* $c(\mathcal{P}) = 747.8$*, that is lower than the costs in Examples 2.2 and 3.3, and can be verified to be optimal.* □

3.3 Correlated Filters

We now consider operator placement when the filters in $\mathcal{F}$ may be correlated, i.e., the selectivity of a filter on a stream may depend on the filters that have already been applied. We define the *conditional selectivity* of a filter F given a set of filters $\mathcal{Q} \subseteq \mathcal{F}$, denoted $s(F|Q)$, as the fraction of tuples that satisfy F given that they satisfy all the filters in $\mathcal{Q}$. Note that if $F \in \mathcal{Q}$, $s(F|Q) = 1$.

When filters are correlated, Theorem 2.1 no longer holds. In fact, the problem of optimal ordering of correlated filters at a single node has been shown to be NP-hard [9, 17]. The same work also gives a natural greedy algorithm based on conditional selectivity (Figure 7) that is guaranteed to find an ordering having a cost at most 4 times the optimal cost. The algorithm defines the conditional rank for each filter (line 3) and at each step, picks the filter having the smallest conditional rank to be executed next. It is also shown that this approximation ratio of 4 is the best possible unless $P = NP$.

Our problem of optimally executing a set of correlated filters at multiple nodes is clearly at least as difficult as the single-node problem, and hence is NP-hard. We show in this section that the same approximation ratio of 4 can be obtained for our problem setting. Since Figure 7 is the best ordering for correlated filters that can be found efficiently, we assume that in any in-network query plan, the set of filters at any node are executed in the order given by Figure 7.

Algorithm ***CORRELATED***
$\mathcal{F}$: Set of correlated filters to be ordered
1. $\mathcal{Q} = \emptyset$
2. while ($\mathcal{Q} \neq \mathcal{F}$)
3. conditional rank$(F) = \frac{\text{cost}(F)}{1-s(F|Q)}$ $\forall F \in \mathcal{F}$
4. $F_{min} = F \in \mathcal{F}$ that has smallest conditional rank
5. choose F_{min} to be executed next; $\mathcal{Q} = \mathcal{Q} \cup \{F_{min}\}$

Figure 7: Ordering of Correlated Filters [9, 17]

We again model the network links as filters with cost and selectivity as before (given by Lemma 3.4). Additionally, the filters that model network links are *independent* of all filters, i.e., for each i, $s(F^l_i|\mathcal{Q}) = s(F^l_i)$ for any $\mathcal{Q}$ such that $F^l_i \notin \mathcal{Q}$. Assume as in Section 3.2 that all ranks are calculated at node N_1. First we show that even in the presence of correlated filters, short-circuiting (Section 3.2.1) is still valid.

Lemma 3.8. *Let filters in* $\mathcal{F}$ *be correlated. If rank*$(F^l_{i-1}) >$ *rank*(F^l_i) *for some* $i \in \{2, \ldots, m-1\}$*, then in the optimal solution* $\mathcal{F}_i = \emptyset$*.*

Proof. Suppose $\mathcal{F}_i \neq \emptyset$. Replace the filters in $\mathcal{F}_i$ by a single filter F having equivalent per-tuple cost and selectivity as the filter sequence at node N_i. Now consider rank(F), which may depend on the filters executed at nodes $N_1, \ldots, N_{i-1}$. Since rank$(F^l_{i-1}) >$ rank(F^l_i), either rank$(F) >$ rank(F^l_i) (in which case move F to N_{i+1}) or rank$(F) <$ rank(F^l_{i-1}) (in which case move F to N_{i-1}). Note that this movement of F does not change the rank of any filter since the ranks of F^l_{i-1} and F^l_i are independent of all filters and the rank of F depends only on the filters executed at nodes $N_1, \ldots, N_{i-1}$ which remain the same. Now using a similar argument as in the proof of Lemma 3.6, we see that this movement of F cannot increase the total cost of the solution. Thus $\mathcal{F}_i = \emptyset$ in the optimal solution. □

Theorem 3.9. *Let filters in* $\mathcal{F}$ *be correlated. Let* $r(\mathcal{F}')$ *denote the ordering of a set of filters* $\mathcal{F}'$ *as obtained by algorithm CORRELATED (Figure 7). With this new interpretation of* $r(\mathcal{F}')$*, algorithm OPT_FILTER (Figure 5) gives a 4-approximation to the optimal operator placement.*

Proof. With the new interpretation of $r(\mathcal{F}')$ it is easy to see that Lemma 3.4 holds in the presence of correlated filters. After short-circuiting, the ranks of the filters $F^l_, \ldots, F^l_{m-1}$ are non-decreasing. Hence when the filters are ordered (by algorithm *CORRELATED*) in line 9 of algorithm *OPT_FILTER*, the filters corresponding to the network links automatically occur in the desired order, and no ordering restrictions on the filters need to be imposed. Since algorithm *CORRELATED* is a 4-approximation to the optimal ordering in the case when there are no ordering restrictions [17], the result follows. □

3.4 Extension to Tree Hierarchies

So far we have restricted our attention to the data acquired by only one of the leaf nodes or sensors of Figure 1. Let S_i denote the stream of data from the ith sensor. We have shown how to optimize the query (1) over any single stream S_i. In reality, query (1) may be posed over data gathered by any number of sensors, i.e., the query is $\sigma_{\mathcal{F}}(S_1 \cup \ldots \cup S_k)$ for k sensors. This query can be written as the union $\sigma_{\mathcal{F}}(S_1) \cup \ldots \cup \sigma_{\mathcal{F}}(S_k)$. Each of the queries in this union operates on different data, so there is no opportunity for sharing computation or transmission among these queries. Hence optimizing their combined execution is equivalent to optimizing each of them separately, for which we use the algorithm of Section 3.2.

4 Joins

Recall the network topology of Figure 2. Now suppose the data acquired by sensor node N_1 is in the form of k different data streams (e.g., a temperature stream, a light stream, a vibration stream, and so on). In this section, we consider in-network processing for queries that involve a sliding-window join [2] of these streams. We assume that the join of all k streams is performed at a single node by the *MJoin* operator [14]; consideration of join trees is left as future work. For ease of presentation, we assume $k = 2$; extension to general k is straightforward.

Let S_1 and S_2 be the streams acquired by sensor node N_1. We consider the query:

$$\texttt{SELECT} * \texttt{FROM} (\texttt{S}_1[\texttt{W}_1] \bowtie \texttt{S}_2[\texttt{W}_2]) \quad \texttt{WHERE } \texttt{F}_1 \wedge \ldots \wedge \texttt{F}_\texttt{n} \tag{7}$$

where W_1 and W_2 represent the lengths of the windows (time-based or tuple-based) on streams S_1 and S_2, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ is a set of filters. We extend the cost model of Section 2.1 to include the selectivity and cost of the join operator.

1. **Selectivity**: The selectivity $s(\bowtie)$ of the join is defined as the fraction of the cross product that occurs in the join result. Thus if streams S_1 and S_2 have rates of r_1 and r_2 tuples per unit time coming into the join operator, the output of the join is at rate $s(\bowtie)r_1r_2$.
2. **Cost**: The cost per unit time of performing the join is given by

$$\text{cost}(\bowtie) = a_1r_1 + a_2r_2 + a_3r_1r_2 \tag{8}$$

where a_1, a_2, and a_3 are constants. This form arises because a constant amount of work must be done per input tuple (therefore $a_1r_1 + a_2r_2$), and similarly a constant amount of work to output every join tuple (therefore $a_3r_1r_2$). Just as the filter costs, the join cost also scales down by a factor γ_i on moving from node N_i to N_{i+1}.

Given cost and selectivity for the join operator, an expression analogous to (3) for the total cost of an in-network query plan for query (7) can be written.

Divide the set of filters $\mathcal{F}$ into $\mathcal{F}^1, \mathcal{F}^2$, and $\mathcal{F}^{1,2}$. For $i = 1, 2$, $\mathcal{F}^i$ consists of those filters that may be applied either on S_i before the join or on the join result after the join (denote $|\mathcal{F}^i|$ by n_i). $\mathcal{F}^{1,2}$ can be applied only on the join result. We assume that the join and the filters are independent, i.e., the selectivity of any operator does not depend on the operators applied earlier. We have the following result (similar to Theorem 5.4 in [3]). We omit the proof since it is mostly an adaptation of the proof in [3] to incorporate network links and cost changes between nodes.

Theorem 4.1. *Given join cost of the form* (8)*, in the optimal in-network query plan for* (7)*, the filters in $\mathcal{F}^1$ (or $\mathcal{F}^2$) must be executed in rank order.* □

Let $F^1_1, \ldots, F^1_{n_1}$ be the filters in $\mathcal{F}^1$ in rank order. By Theorem 4.1, in the optimal plan there exists an i such that first $F^1_1, \ldots, F^1_i$ are executed on stream S_1, followed by the join, and then $F^1_{i+1}, \ldots, F^1_{n_1}$ on the join result (similarly for $\mathcal{F}^2$). Additionally, the join

Algorithm ***OPT_FILTER_JOIN***
1. $\mathcal{P}^* = \text{NULL}, c(\mathcal{P}^*) = \infty$
2. for $i = 0$ to n_1, $j = 0$ to n_2, $k = 1$ to m
3. Construct new $\mathcal{P}$ with join at node N_k
4. Optimally place $F_1^1, \ldots, F_i^1$ at nodes $N_1, \ldots, N_k$
5. Optimally place $F_1^2, \ldots, F_j^2$ at nodes $N_1, \ldots, N_k$
6. Optimally place remaining filters at $N_k, \ldots, N_m$
7. if $\big(c(\mathcal{P}) < c(\mathcal{P}^*)\big)$ $\mathcal{P}^* = \mathcal{P}$
8. return $\mathcal{P}^*$

Figure 8: Operator Placement for Queries with Joins

can be executed at each of the m nodes. Our algorithm (Figure 8) finds the optimal plan by an exhaustive search through these options. Lines 4-6 are each an invocation of algorithm *OPT_FILTER*. Hence the algorithm is polynomial, and a simple implementation runs in $O(n_1 n_2 m(n+m)log(n+m))$ time.

5 Extensions

In this section, we define some interesting variations of the basic filter placement problem as future work, and we demonstrate their hardness.

5.1 Constrained Nodes

In some scenarios we may have constraints on the total amount of filter execution and transmission cost that certain nodes can incur. Given cost constraints at each node, a new problem is to find a feasible in-network query plan that satisfies these constraints, if such a plan exists. Recall the definition of $c(\mathcal{P}, i)$ from Section 3.1.

Definition 5.1 (Feasible Operator Placement). *Given cost constraint C_i at node N_i, find an in-network query plan $\mathcal{P}$ such that $c(\mathcal{P}, i) \leq C_i$ for each i or return* `NO` *is no such $\mathcal{P}$ exists.* □

Theorem 5.2. *The feasible operator placement problem is NP-hard.*

Proof. By reduction from *PARTITION* [11]. □

In cost-constrained environments, a further desirable property might be load balancing: We might prefer a plan having overall higher cost if it places roughly equal load on each node, as compared to a plan that has lower cost but loads a few nodes very heavily or up to capacity. Load balancing may be particularly applicable when the system is required to support a number of concurrent queries.

Definition 5.3 (Load-Balancing). *Given cost constraint C_i at node N_i, find the in-network query plan $\mathcal{P}$ that minimizes $\max_{1 \leq i \leq m}\{c(\mathcal{P}, i)/C_i\}$.* □

Clearly, the load balancing problem is at least as hard as the feasible operator placement problem, since if $\max\{c(\mathcal{P}, i)/C_i\} < 1$ we have found a feasible operator placement. Thus, the load balancing problem is NP-hard.

5.2 Per-Filter Cost Scaling

So far we have assumed that the cost of each filter scales down by a factor γ_i from node N_i to N_{i+1}. However, the cost of different filters may change differently from one node to the next, i.e., $c(F, i+1)/c(F, i)$ may be different for different F. For example, if a filter F accesses external data that resides close to node N_i, it may be more expensive to execute F at node N_{i+1} than at node N_i. Meanwhile, other filters may be cheaper at node N_{i+1} simply because N_{i+1} has higher computational power. When we have per-filter cost scaling, the technique we used of modeling network links as filters no longer applies. Whether the problem becomes NP-hard with per-filter cost scaling remains an open question.

6 Related Work

A considerable amount of work has focused on extending centralized data stream query processing systems [2] to a distributed setting, e.g., Borealis [1], HourGlass [18], IrisNet [7], and NiagaraCQ [4]. Most of this work considers internet-style network topologies consisting of nodes with ample computational power. Consequently, the work focuses on optimizing network usage and minimizing latency, and is not concerned with computational overload. Even when computational overload is considered, e.g, in [5], only heuristics are provided to move load from one node to another.

Our paper addresses the considerably different scenario of data acquisition environments [10], where optimization of both communication and computation is required. There has been some previous work on in-network processing in these environ-

ments, but it focusses primarily on aggregation [15], and has not considered expensive filters or joins. Acquisitional query processing [16] focusses on where, when, and how often data is physically acquired and delivered to the query operators, but the problem of operator placement is not dealt with.

In classical relational query optimization, filters are usually assumed to be inexpensive, and a common optimization heuristic is to push filters as close to the leaf operators as possible. Query optimization and site selection for distributed databases is also well studied [8, 12], again assuming inexpensive filters. Expensive filters have been considered in the context of query optimization with user-defined predicates [3, 13], but only in a centralized setting.

7 Conclusions

This paper addresses the problem of query processing in sensor data environments with progressively increasing computational power and network bandwidth up a hierarchy of processing nodes. Data is acquired at low-capability edge devices and transmitted up the hierarchy to the root, where queries are posed and results collected. To reduce bandwidth, query operators can be executed lower in the hierarchy, typically at the expense of higher computational cost. We address the problem of balancing computational cost against network bandwidth to obtain an optimal operator placement algorithm that minimizes overall cost. We show that the problem is tractable, but that a greedy algorithm can be suboptimal. We provide an optimal algorithm for uncorrelated filters, then extend our approach to correlated filters and multiway stream joins. Finally, we pose related open problems for future research.

References

[1] Y. Ahmad and U. Cetintemel. Network-aware query processing for stream-based applications. In *Proc. of the 2004 Intl. Conf. on Very Large Data Bases*, pages 456–467, Sept. 2004.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the 2002 ACM Symp. on Principles of Database Systems*, pages 1–16, June 2002.

[3] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Trans. on Database Systems*, 24(2):177–228, 1999.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–390, May 2000.

[5] M. Cherniack et al. Scalable distributed stream processing. In *Proc. First Biennial Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[6] C. Cranor et al. Gigascope: high performance network monitoring with an SQL interface. In *Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, page 623, May 2002.

[7] A. Deshpande, S. Nath, P. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 503–514, 2003.

[8] R. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational data base system. In *Proc. of the 1978 ACM SIGMOD Intl. Conf. on Management of Data*, pages 169–180, May 1978.

[9] U. Feige, L. Lovász, and P. Tetali. Approximating min-sum set cover. *Algorithmica*, 2004.

[10] M. Franklin et al. Design Considerations for High Fan-in Systems: The HiFi Approach. In *Proc. Second Biennial Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2005. (To appear).

[11] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[12] L. Haas et al. R*: A Research Project on Distributed Relational DBMS. *IEEE Data Engineering Bulletin*, 5(4):28–32, 1982.

[13] J. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proc. of the 1993 ACM SIGMOD Intl. Conf. on Management of Data*, pages 267–276, 1993.

[14] J. Kang, J. F. Naughton, and S. Viglas. Evaluating window joins over unbounded streams. In *Proc. of the 2003 Intl. Conf. on Data Engineering*, Mar. 2003.

[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th USENIX Symposium on OSDI*, Dec. 2002.

[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 491–502, 2003.

[17] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. *Proc. Intl. Conf. Database Theory*, 2005. (To appear).

[18] P. Pietzuch et al. Path optimization in stream-based overlay networks. Technical report, Harvard University, 2004.