Power-Aware Operator Placement and Broadcasting of Continuous Query Results [1]

Panayiotis Neophytou, Mohamed A. Sharaf, Panos K. Chrysanthis, Alexandros Labrinidis
Department of Computer Science, University of Pittsburgh
{panickos, msharaf, panos, labrinid}@cs.pitt.edu

ABSTRACT

Complex event detection over data streams has become ubiquitous through the widespread use of sensors, wireless connectivity and the wide variety of end-user mobile devices. Typically, such event detection is carried out by a data stream management server executing continuous queries, previously submitted by the users. In this paper, we consider the situation where the end-users submit queries from hand-held devices and the results of the continuous queries, which are in the form of individual data streams, are disseminated to the users over a shared broadcast medium. Specifically, we propose three power-aware query operator placement algorithms that determine which part of a continuous query plan is executed at the data stream management server and which part is executed at the users' wireless device. The algorithms' effectiveness with respect to energy consumption is evaluated using simulation.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing, Distributed Databases*

General Terms

Algorithms, Experimentation, Measurement, Performance.

1. INTRODUCTION

Complex event detection over data streams in monitoring applications has become ubiquitous through the wide spread use of sensors, wireless connectivity and the wide variety of end-user mobile devices. Typically, such event detection is carried out by a *data stream management server* (DSMS) executing *continuous queries* (CQ), that have been previously submitted by the users [23, 2]. In DSMSs, monitoring applications register CQs which continuously process unbounded data streams, looking for data that represent events of interest to the end-user, who can be stationary or mobile. Examples of such applications, of equal interest to stationary and mobile users, include monitoring of stock quotes, airline schedule updates, local news, weather readings, and traffic information. Specifically, we consider applications where wireless connectivity and timeliness of data delivery is inherent, such as emergency management in disaster areas. Another example is the query of a trader (Figure 1) which processes stock price updates, at the floor of a market exchange. First, it filters out stocks that are not in the NASDAQ index using the select operator O_1. Then, it drops columns of no interest (such as source information etc.) using the project operator O_2. Then, it joins the tuples with the user's portfolio to append the buying price using the join table operator O_3. Finally, it calculates the user's profit in the last 5 minutes, every 30 seconds, using the aggregate operator O_4.

In our focused applications, the end-users typically submit queries from hand-held battery-operated devices and receive the results of CQs over a shared wireless medium. A DSMS could be effortlessly integrated with any of the existing wireless broadcast communication infrastructure (such as MBMS [17], ISDB-T [16], StarBand and Hughes Network) to support the continuous dissemination of the results of CQs to the end-users. However, the design of current data dissemination techniques used in wireless networks, especially *data broadcasting*, needs re-thinking because the continuous dissemination of the results clearly stresses the battery capacity as well as the wireless bandwidth.

Our goal in this work, is to design operator placement algorithms that work in synergy with the broadcast organization so that to minimize the total energy consumption on the hand-held devices. Our approach is based on the observation that the energy costs for receiving data are typically much higher than the energy costs for processing that data, an observation that has also motivated in-network processing in wireless sensor networks (e.g., [14, 20]). Hence, it is often more beneficial to broadcast an intermediate query result with a relatively smaller size than to broadcast the final result. At the same time the set of query operators processing the intermediate result to produce the final result, is shipped to the user-side. In the example described above, the first two operators O_1 (Select) and O_2 (Project) are data reducing, while the last two operators O_3 (Join Table) and O_4 (Aggregation) could potentially be data expanding. Thus, O_3 and O_4 could be shipped to the wireless user, and the much smaller intermediate result produced by O_2 will be broadcast. The join operator, O_3 can be processed on the client, since the joining portfolio table is user-specific. However, this operator shipping will incur an extra processing energy cost at the user device.

Thus, the key challenge in designing an effective operator placement scheme is the balancing of the trade-off between the reduction in tuning energy and the increase in processing energy so as to minimize the overall energy consumption at the end-user devices.

[1]This research was supported in part by NSF grant IIS-0534531 and NSF career award IIS-0746696

MobiDE'10, June 6, 2010, Indianapolis, Indiana, USA.

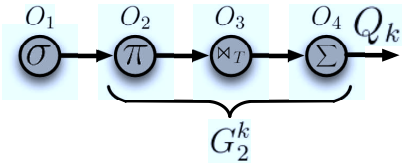

Figure 1: Continuous Query Plan

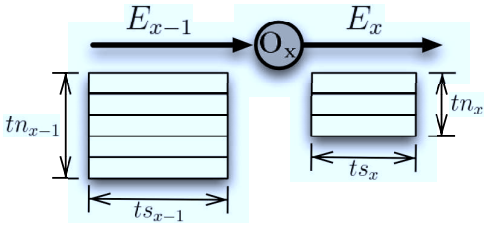

Figure 2: Operator model

Towards this end we have developed three operator placement algorithms that use different criteria for selecting the operators to be placed on the user device. In particular, we propose *MinDataCut* which focuses on minimizing the tuning energy; *MinPowerCut* which extends MinDataCut to consider the processing energy needed for executing operators at the client-side; and *BOSe* which further extends the previous two algorithms to also consider the broadcast data organization. In the context of this paper, we optimize only for energy without considering any other metrics such as response time minimizing which was the goal of existing operator placement schemes in Distributed DSMSs ([3]). Also, we consider CQs similar to the above example with *select*, *project*, *aggregate*, and *join-table* operators.

The contributions of this paper are summarized as follows:

1. We present three algorithms for the placement of continuous query operators across the DSMS and mobile clients in a wireless data broadcasting environment that result in differing degrees of energy savings.
2. We integrate our proposed algorithms with both sorted and indexed data broadcast organizations.
3. We provide an extensive experimental evaluation of our proposed algorithms under different settings for sensitivity analysis. Our results show that BOSe performs best and can achieve an overall energy reduction of up to 53%.

Roadmap: Section 2 provides the system model. Our three operator placement algorithms are presented in Section 3. Section 4 describes our experimental setup and discusses the experiments and results. Section 5 surveys related work.

2. SYSTEM MODEL

In our model, a DSMS allows wireless users to register CQs. In addition to the standard modules of CQ admission manager, query optimizer, CQ scheduler, memory manager and load shedder, the DSMS implements a wireless disseminator which broadcasts the results of CQ to the wireless users.

2.1 Data Stream Processing

A CQ evaluation plan generated by the query optimizer can be conceptualized as a data flow tree [2, 9], where the nodes are operators that process tuples and edges represent the flow of tuples from one operator to another (Figure 1). An edge from operator O_x to operator O_y means that the output of O_x is an input to O_y. Hence, $upstream(O_y) = O_x$ and $downstream(O_x) = O_y$. Each operator is associated with a *queue* where input tuples are buffered until they are processed.

A *single-stream query* Q_k has a single *leaf* operator O^k_{leaf} and a single *root* operator O^k_{root}. For example, in Figure 1, O_1 is the leaf, whereas O_4 is the root. Further, in a query plan Q_k, an *operator segment* $G^k_{x,y}$ is the sequence of operators that starts at O^k_x and ends at O^k_y. If the last operator on $G^k_{x,y}$ is the root operator, then we simply denote that operator segment as G^k_x. Figure 1 shows an operator segment $G^k_2 = < O_2, O_3, O_4 >$.

In a query, each operator O^k_x (or simply O_x) could be one of four types: select (σ), project (π), aggregate (e.g. $\sum$), or join-table ($\bowtie_T$). Each operator is associated with three parameters:

1. **Processing cost** (c_x): is the number of cycles needed to process an input tuple.
2. **Selectivity** or **Productivity** (s_x): is the ratio of output tuples produced by O_x after processing one input tuple. Thus, s_x is less than or equal to 1 for a filter operator and it could be greater than 1 for a join operator.
3. **Projectivity** (p_x): is the ratio between the size of a tuple produced by O_x (i.e., output size) to its size before being processed (i.e., input size). Thus, p_x is less than or equal to 1 for a project operator and it may be greater than 1 for a join operator.

For an operator O_x, with $upstream(O_x) = O_{x-1}$, we define the following characterizing parameters (Figure 2):

- **Number of Output Tuples (tn_x):** is the number of tuples produced at the output of O_x after processing the tn_{x-1} tuples in its input queue.

$$tn_x = tn_{x-1} \times s_x$$

- **Size of Output Tuples (ts_x):** is the size of each tuple produced at the output of O_x after processing a tuple in its input queue.

$$ts_x = ts_{x-1} \times p_x$$

- **Size of Output Data (ds_x):** is the size of the data block produced at the output queue of O_x after processing a block of data from its input queue.

$$ds_x = tn_x \times ts_x$$

Notice that if O_x is the root operator in query Q_k (i.e., $O_x = O^k_{root}$), then ds_x is the total size of the data block produced by Q_k.

2.2 Wireless Broadcast

In wireless networks, broadcasting is the natural method to propagate information and guarantee scalability for bulk data transfer. Specifically, data can be efficiently disseminated by any combination of the following two schemes: *broadcast push* and *broadcast pull* [5, 24].

In this work, we adopt broadcast push since it naturally complies with the DSMS access model where a client installs a CQ once and the server repeatedly broadcasts the new results as they become available. Hence, any number of clients can monitor the broadcast channel and retrieve data as it arrives, at a constant bandwidth speed: BW.

In our model, the *wireless disseminator* initiates a new broadcast cycle as soon as the previous one ends and consists of the results produced by the CQs during the previous broadcast cycle. We denote the broadcast result of query Q_k as D_k and its size in bytes as $\mid D_k \mid$. If the final result of a query is broadcast, i.e., the result produced at the query root operator, then $\mid D_k \mid = ds^k_{root}$. Otherwise, if the output of an internal operator O^k_x is broadcast, then $\mid D_k \mid = ds^k_x$.

The result D_k of each query Q_k appears on the broadcast channel as a contiguous sequence of data packets preceded by a descriptor packet that contains an identifier of Q_k and the time offset to the next broadcast cycle. Accordingly, each client should *tune* to each broadcast cycle for results corresponding to its registered CQs. During tuning, the client's network interface card (NIC) is in *active mode* consuming relatively large amounts of energy compared when the client's NIC is switched to an *idle mode*. Hence, the amount of energy consumed by a wireless client depends on the data organization [4, 13, 12].

In this paper, we adopt two basic data organization schemes. The first one uses *sorted broadcast* where the broadcast server sorts the query results according to data size. In particular, each query result D_k is assigned a priority equal to $\frac{1}{|D_k|}$ and the broadcast is organized in descending order of priority. Hence, results with smaller size (or equivalently, shorter transmission time) will appear first on the broadcast. This clearly maintains a broadcast that follows the *shortest job first* scheduling policy which has been shown to minimize total response time in on-demand data broadcast [1]. Accordingly, for a client N_i that registered query Q_k, its tuning time $T_T(N_i)$ is computed as:

$$T_T(N_i) = \frac{\mid D_k \mid + \sum_i \mid D_i \mid}{BW}, \text{where } \forall i : \mid D_i \mid < \mid D_k \mid$$

The second data organization scheme uses an *indexed broadcast* where the broadcast server attaches an index at the beginning of each broadcast cycle (i.e., a (1,1) index [13]). The index contains an entry for each query Q_k on the broadcast in the form $< Q_k, t_k >$, where t_k is the time offset of Q_k's result within the broadcast cycle. Under such a scheme, the client needs to first tune to the index packet to learn the broadcast time t_k of its registered query, then power off the NIC until time t_k. At time t_k, it powers on its NIC again, tune into the broadcast to retrieve the query result (i.e, D_k) and after it finishes fetching all of D_k's data packets, powers off the NIC again until the next broadcast cycle. The tuning time for a node N_i in the indexed broadcast is computed as:

$$T_T(N_i) = \frac{\mid D_k \mid + \mid Index \mid}{BW}$$

where $\mid Index \mid$ is the size of the index, which could change from one cycle to another depending on the number of queries whose results are disseminated in the broadcast.

3. OPERATOR PLACEMENT

In this section, we formalize the placement of CQ operators in wireless DSMS environments and propose three operator placement algorithms, namely: *MinDataCut*, *MinPowerCut*, and *BOSe*.

3.1 Problem Statement

Our goal is to design operator placement algorithms that work in synergy with the broadcast organization so that we minimize the total energy consumption at the wireless clients. The total energy consumption is the sum of two components: *tuning* and *processing*, which can be expressed as:

$$E_{Total} = E_{Tune} + E_{Process} \quad (1)$$

As discussed in the introduction, our approach to minimizing Eq.1 is based on the observation that the energy costs for receiving data is typically much higher than the energy costs for processing that data. Hence, it is often beneficial to broadcast an intermediate query result with a relatively smaller size than to broadcast the final result. Accordingly, the set of query operators processing that intermediate results to produce the final result are moved to the client-side incurring an extra processing energy.

For the algorithms to be able to calculate the trade-offs between tuning and processing energy for each client, when a client registers a CQ for processing it also attaches its profile. This profile includes three parameters regarding the client's characteristics:

1. $Speed(N_i)$: the processing speed of the client in cycles per unit of time.
2. $P_P(N_i)$: the power consumed per unit of time of processing.
3. $P_T(N_i)$: the power consumed per unit of time when the NIC is active.

Hence, the tuning energy E_{Tune} consumed by each wireless client N_i is expressed as:

$$E_{Tune}(N_i) = T_T(N_i) \times P_T(N_i) + U(N_i) \times E_{PowerUp}$$

where $T_T(N_i)$ is the tuning time, $U(N_i)$ is the number of times the client needs to power up the NIC and $E_{PowerUp}$ is the amount of energy incurred in powering up the NIC.

Given the clients profiles and their corresponding registered queries, an operator placement algorithm decides to shift some of the computation to the client-side, if it is beneficial in reducing the overall total energy consumption. Specifically, it splits the query plan into two segments, where the first segment is processed on the server, whereas the second segment is processed on the client. For instance, if client N_i registered query Q^i, then Q^i might be split at operator O^i_x with segment $G^i_{leaf,x}$ on the server and $G^i_{x+1,root}$ on the client.

For a client N_i running segment $G^i_{x+1,root}$, the processing time is computed as:

$$T_P(N_i) = \sum_{O_j \in G^i_{x+1,root}} \frac{c_j \times tn_{j-1}}{Speed(N_i)}$$

Hence, the processing energy $E_{Process}$ consumed by each wireless client N_i is expressed as:

$$E_{Process}(N_i) = T_P(N_i) \times P_P(N_i)$$

3.2 MinDataCut

In this first algorithm, our objective is to focus on the tuning energy component in Eq.1. That is, minimize the total amount of energy spent by the wireless clients for tuning into the broadcast channel. Towards this, we leverage the discrepancies in size between the intermediate results within each query. To illustrate this, assume a query Q_k composed of operators $< O^k_l, ..., O^k_x, O^k_y, ..., O^k_r >$ where the sizes of the intermediate results are expressed as: $< ds^k_l, ..., ds^k_x, ds^k_y, ..., ds^k_r >$. This sequence of data sizes is typically non-monotonic as it might increase or decrease across segments of operators. For example, if $(s^i_y \times p^i_y) < 1$, then $ds^k_y < ds^k_x$. That is, O^k_y is a data reduction operator which decreases the size of its input data. On the other hand, if $(s^i_y \times p^i_y) > 1$, then $ds^k_y > ds^k_x$. That is, O^k_y is a data production operator which increases the size of its input data.

To minimize the tuning energy of query Q_k, it is sufficient to select the intermediate result with the smallest size, say ds^k_x. This

result is selected for broadcast and all the operators following O_x are shipped to the client. In the general case, however, our goal is to minimize the total energy. That is, $\sum_{i=0}^{m} E_{Tune}(N_i)$, where m is the number of clients/queries. For this, it is sufficient to find the set of edges $\mathbb{E}$ in the operator plan that satisfy the following conditions:

1. Each query Q_i has exactly one edge E_x^i in $\mathbb{E}$, and
2. $\sum_{i=0}^{m} \frac{ds_x^i}{BW}$ is minimum.

The first condition above ensures that there exists one intermediate result for each query, hence the final result of each query could be produced at the client side after shipping the necessary operators. The second condition ensures that the total size of those intermediate results is minimum.

Once the set of edges $\mathbb{E}$ is identified, we start moving operators to the client side. In particular, for a query Q_i, if $E_x^i \in \mathbb{E}$, then all the operators following operator O_x^i are moved to the client side. That is, $<O_{x+1}^i, ..., O_r^i>$.

3.3 MinPowerCut

As discussed above, MinDataCut is expected to reduce the total tuning energy since it selects the broadcast with the minimum length. MinDataCut, however, is oblivious to the clients capabilities and limitations, such as processing speed and power. Thus, if the operators to the right of the edges in $\mathbb{E}$ are expensive to process, there is a possibility that the overall total power consumption (i.e., tuning and processing) will be even higher than processing everything at the server.

To avoid the drawback of MinDataCut, a natural extension is to label the edges according to both the processing energy cost as well as the data tuning energy cost. Specifically, the processing cost is the expected time needed to process the intermediate result at the client side. For instance, in query Q_k, an edge E_x^k will be labeled with ds_x as before, in addition to the cost of processing the output of O_x (i.e., tn_x) by the operators segment G_{x+1}^i, which is the segment of operators that starts after the min-cut point and ends at the root. Thus, the edge selection algorithm must satisfy the following conditions:

1. Each query Q_i has exactly one edge E_x^i in $\mathbb{E}$, and
2. $\sum_{i=0}^{m} \left(\frac{ds_x^i}{BW} \times P_T(N_i) + \sum_{O_j \in G_{x+1}^i} \frac{tn_{j-1} \times c_j}{Speed(N_i)} \times P_P(N_i) \right)$ is minimum.

This is achieved by selecting the edge with the minimal label from each query.

3.4 Broadcast-Aware Operator Selection

MinPowerCut as opposed to MinDataCut, which only considers the tuning energy, provides the advantage of considering the energy needed for processing at the client side. This will usually lead MinPowerCut to select an edge towards the end of the query plan since it will involve placing less operators at the client. However, the selected edge by MinPowerCut might have a higher data size than the edge selected by MinDataCut. As a result, the length of the broadcast cycle generated by MinPowerCut is expected to be higher than MinDataCut. Under a broadcast data dissemination mode, increasing the length of the broadcast cycle has a negative impact on *all* clients. In particular, each client will have to wait longer to receive its query result and in the case of sorted broadcast, spending more tuning energy while waiting.

The *Broadcast-Aware Operator Selection (BOSe)* balances the trade-off between tuning energy and processing energy. BOSe could be perceived as a hybrid of MinDataCut and MinPowerCut as it integrates the desirable features of each. On one hand, like MinDataCut, BOSe tries to minimize the length of the broadcast cycle so that to minimize tuning energy. On the other hard, BOSe, like MinPowerCut, considers the extra energy needed for operator processing at the client side.

BOSe uses the MinDataCut output as a starting point and then applies a greedy selection process geared towards finding a segment of operators within each query and reinstate them back on the server. Since MinDataCut gives the minimal broadcast size, that means that any reinstatement by BOSe will incur an increase in the broadcast no matter what. Thus, BOSe will only perform a reinstatement if its *benefit* in terms of reducing processing energy is greater than the *cost* incurred in terms of increasing tuning energy, which depends on the broadcast organization.

At each iteration, BOSe examines all the current edges (i.e., $\mathbb{E}$) between the server and clients. For each edge $E_x \in \mathbb{E}$, it generates a list of all the possible segments of operators following that edge. That is, all the prefixes of the operator segment $G_{x+1,root}$. For instance, in $<O_1^k, O_2^k, O_3^k, O_4^k>$, if $E_1^k \in \mathbb{E}$, then BOSe will consider the cost/benefit of reinstating any of the segments: $< O_2^k >$, $< O_2^k, O_3^k >$, or $< O_2^k, O_3^k, O_4^k >$ back on the server. This process is performed for each edge in $\mathbb{E}$ and the segment with highest impact in reducing total energy is selected and its operators are reinstated to the server. BOSe repeats the selection process until no further improvement in energy is achievable.

3.4.1 BOSe optimization function

Recall that the total energy spent by wireless clients is the sum of two components: tuning and processing as expressed in Eq. (1). Also recall that at each step, BOSe is expected to increase the tuning energy while decreasing the processing energy as compared to MinDataCut. Assume these changes are Δ_{Tune} and $\Delta_{Process}$, respectively. Hence, after BOSe makes a selection, the new overall energy consumption can be expressed as:

$$E_{Total} = (E_{Tune} + \Delta_{Tune}) + (E_{Process} - \Delta_{Process})$$

Clearly, our objective is to select an operator segment which minimizes the value: $\Delta_{Tune} - \Delta_{Process}$. Thus, we simply need to compute that value for each operator segment under consideration and select the one with the lowest value. To illustrate this process, assume that $E_R \in \mathbb{E}$, hence, BOSe needs to examine all the prefixes of operator segment $G_{R+1,root}$. Further, assume that G is one of those prefixes under examinations, which is considered as a candidate to be moved from the client N_i back to the server side. For instance, in Figure 3, $G =< O_2^2, O_3^2 >$. Moving G back to the server will reduce the processing energy by the amount $\Delta_{Process}$, which is computed as follows:

$$\Delta_{Process} = \sum_{O_j \in G} \frac{tn_{j-1} \times c_j}{Speed(N_i)} \times P_P(N_i)$$

Further, moving segment G back to server entails removing its input edge from the broadcast (i.e., edge E_R). Further, it also entails replacing E_R with G's output edge, say E_A. That is, E_R is removed from $\mathbb{E}$, whereas E_A is added to $\mathbb{E}$ (Figure 4). Hence, we need to compute the impact of that remove/addition operation on the tuning energy (Δ_{Tune}) which depends on the broadcast organization.

In the case of *sorted broadcast*, removing E_R will reduce the tuning time for the client N_i receiving data from E_R. Addition-

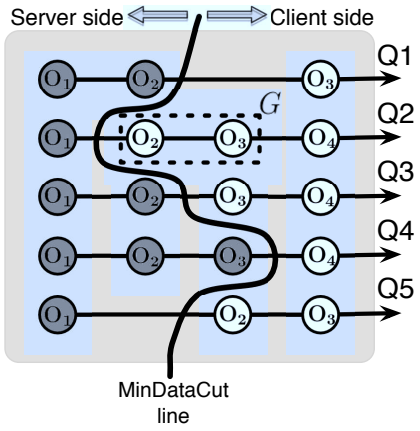

Figure 3: MinDataCut and operator segment $G_{2,3}$.

ally, it will also reduce the tuning time for all the clients waiting for results that appear after E_R on the broadcast cycle. This reduction per client is simply ds_R/BW, where ds_R is the data size of associated with edge E_R. Hence, the total reduction is computed as:

$$\Delta_R = \frac{ds_R}{BW} \times (P_T(N_i) + \sum_{N \in \mathbb{N}_\mathbb{R}} P_T(N)) \quad (2)$$

where $\mathbb{N}_\mathbb{R}$ is the set of clients waiting for results that appear after E_R on the broadcast cycle.

Similarly, adding E_A will increase the tuning time for the client N_i receiving data from E_A. Additionally, it will also increase the tuning time for all the clients waiting for results that appear after E_A on the broadcast cycle (Figure 4). This increase per client is simply ds_A/BW, where ds_A is the data size of associated with edge E_A. Hence, the total increase is computed as:

$$\Delta_A = \frac{ds_A}{BW} \times (P_T(N_i) + \sum_{N \in \mathbb{N}_\mathbb{A}} P_T(N)) \quad (3)$$

where $\mathbb{N}_\mathbb{A}$ is the set of clients waiting for results that appear after E_A on the broadcast cycle.

Moreover, under a sorted broadcast, the location of the new edge E_A on the broadcast cycle is determined according to its data size. Since $ds_A > ds_R$, then E_A will appear at a further offset than the one where E_R. This entails that when E_A replaces E_R, the client N_i will have to spend more tuning time to receive E_A than what it used to spend to receive E_R. This translates into extra tuning energy which is computed as:

$$\Delta_{R,A} = \frac{\sum_{E \in \mathbb{E}_{\mathbb{R},\mathbb{A}}} ds_E}{BW} \times P_T(N_i) \quad (4)$$

where $\mathbb{E}_{\mathbb{R},\mathbb{A}}$ is the set of edges on the broadcast cycle that appear after E_R and before E_A.

Thus, Δ_{Tune} for the sorted broadcast is computed as:

$$\Delta_{Tune} = -\Delta_R + \Delta_A + \Delta_{R,A}$$

In the case of *indexed broadcast* the optimization is simplified since the edge remove/add operation will only affect the client under examination without any impact on the other clients in the system (i.e., $\Delta_{R,A} = 0$ in Eq. 4). Thus, Δ_{Tune} for the indexed broadcast is denoted by $\Delta'_{Tune} = -\Delta'_R + \Delta'_A$, where equations 2 and 3 become

$$\Delta'_R = \frac{ds_R}{BW} \times P_T(N_i)$$

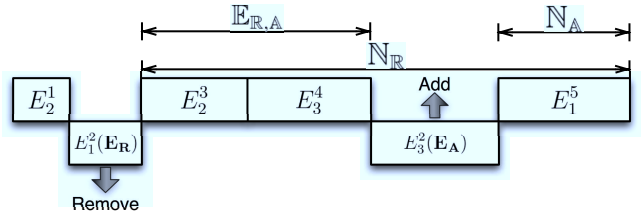

Figure 4: Optimization step for BOSe algorithm

$$\Delta'_A = \frac{ds_A}{BW} \times P_T(N_i)$$

It is interesting to note that the above equations clearly reveal that BOSe on indexed broadcast behaves like MinPowerCut on indexed broadcast. This fact is also confirmed by our experimental results in Section 4.

4. EXPERIMENTAL EVALUATION

In this section, we illustrate the performance of our proposed algorithms over various parameters using a homegrown simulator. First we present the experimental setup, and then we present and discuss the results.

4.1 Experimental Setup

The input of our simulator is a complete plan of all the registered CQs, along with the clients' profiles which include their speed and power consumption parameters for processing and tuning. Each CQ consists of a set of operators and edges as described in our model (Section 2.1), and each query is registered by one wireless client.

The algorithms are implemented in Java and run as they would on a real system as long as the query plan is modeled using the data model used in the code. We have implemented our algorithms, namely: *MinDataCut*, *MinPowerCut*, and *BOSe*. We have also implemented an additional algorithm, *ServerOps*, which is our base case, as it executes all the operators on the server and broadcasts the final results to the clients.

For each algorithm, we have measured its performance in terms of average energy consumption per client, which includes tuning and processing. The parameters used for each set of experiments are summarized in Table 1. In each experiment, we vary one of these parameters for sensitivity analysis, as described in Section 4.2.

Table 1: Workload Default Characteristics

Parameter	Values
Number of queries	20 – 300 (default 50)
Levels per query	2 – 20 (default 10)
Sources tuple rate	500 – 1000 tuples/sec
Sources tuple size	2000 – 4000 bytes
Selectivity	0.2 – 1.8, uniform
Projectivity	0.5 – 1.5, uniform
Operator costs	$100 \times 10^6 - 200 \times 10^6$ cycles
Operator cost skewness	0.2 increments (Zipf)
Hand-held device speed	1×10^9 cycles/sec
Bandwidth	125000 bytes/sec
Processing vs. Tuning power	0.16

The number of levels per query refers to the number of operators which exist in every single-stream query in the workload. Selectivities and projectivities are uniformly assigned across operators

according to the values mentioned in Table 1. The cost of all the operators that are at the same level across queries is generated according to a Zipf distribution per level. The skewness of the Zipf distribution per level is towards the high cost and is proportional to the level number; in the default setting the skewness of operator level i is equal to $0.2 \times i$. The processing vs. tuning power refers to the ratio between the power needed per unit of time of query processing on the wireless client vs. the power needed by the same client for a unit of time of tuning into the broadcast network. We chose that ratio to be 0.16, but we also perform a sensitivity analysis on this parameter as discussed below. The results reported are in simulated units of energy according to the energy model described in Section 2.

4.2 Results

We present the performance results under the settings shown in Table 1. The values reported are averages of 15 runs for each experimental setting. We conducted the experiment using both sorted broadcast as well as indexed broadcast.

4.2.1 Processing over tuning energy cost ratio

In our first experiment, we measured the total energy consumption at all the wireless clients, as the ratio of processing to tuning power of the nodes increases linearly from 0.01 to 0.2, while keeping the tuning power constant.

Sorted Broadcast: Figure 5 shows that the energy performance of *ServerOps* is constant because this algorithm runs all operators at the server side, hence, increasing the processing power of wireless clients will have no impact on its performance. The figure also shows that *MinDataCut* is linearly increasing. This is because MinDataCut tries to minimize only the tuning energy, but not the processing energy. Hence, *MinDataCut* selects the same set of edges for every setting regardless of the increase in power consumption needed for processing. As the power ratio linearly increases, the total power needed by MinDataCut also increases linearly. Once the processing power becomes high enough, its performance becomes even worse than that of ServerOps because the increase in processing costs at the nodes dwarfs the savings gained from just minimizing the broadcast size. *MinPowerCut* performs better than MinDataCut and ServerOps for higher processing to tuning power ratios, but it is worse than MinDataCut until the 0.08 ratio point, because up to that point the tuning energy consumption would still dominate the processing one. MinPowerCut, however, is oblivious to the broadcast organization as it considers each query individually without measuring the impact of its selected edge on the other clients in the system. For this reason, *BOSe* outperforms MinPowerCut under the sorted broadcast organization as shown in Figure 5. In fact, *BOSe* is always performing better than any of the three other algorithms because it evaluates the different options taking into account both the broadcast organization and the processing power costs of operators running on the wireless clients, thus striking a fine balance between both tuning and processing energies. For instance, at a processing to tuning energy ratio of 0.01 BOSe provides an improvement in energy of 48% over ServerOps and at 0.2 an improvement of 21%. The improvement over MinPowerCut at that point is 4%.

Indexed Broadcast: When using an indexed broadcast *MinDataCut* performs worse than ServerOps even earlier than before because, under indexed broadcast, a client tunes only to its result and the presence of other results on the broadcast has no impact on its tuning energy consumption. Hence, the tuning energy component carries much less weight in the overall energy consumption compared to its counterpart under the sorted broadcast which makes the processing component dominate the energy as early as the 0.04 ratio point as shown in Figure 6. Also, due to the selective tuning provided by indexing, *MinPowerCut* is optimal since the local decisions it makes per query lead directly to a global optimization. That is, the optimization problem is reduced to simply finding the edge with the minimum energy consumption per query. Under this setting, *BOSe* will reinstate operators to the server side until it reaches the same exact results as *MinPowerCut*. For instance, at 0.01 ratio the improvement of both BOSe and MinPowerCut over ServerOps is 53% and at 0.2 it is 7%.

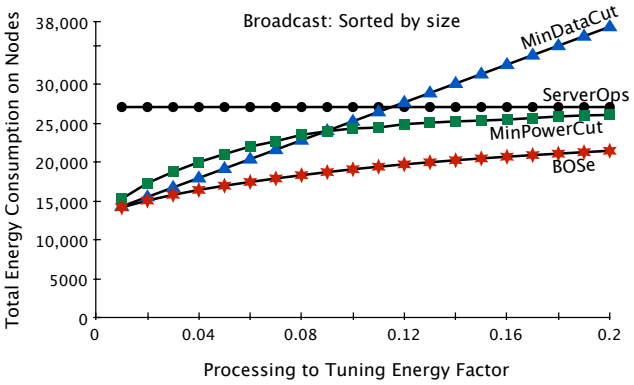

Figure 5: Energy consumption for Tuning Vs Processing per unit of time for sorted broadcast.

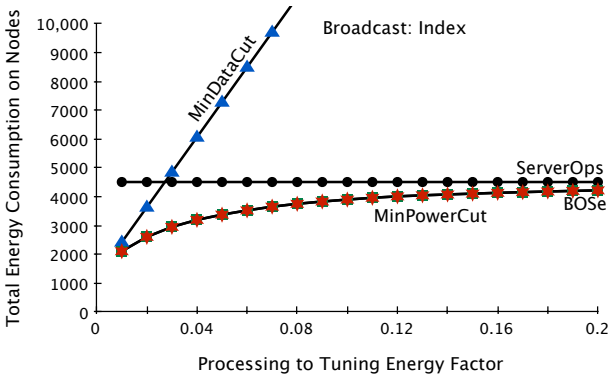

Figure 6: Energy consumption for Tuning Vs Processing per unit of time for indexed broadcast.

Similar to this experiment, we also wanted to examine the sensitivity to the operator cost parameter. Hence, we varied the minimum operator cost in the range between 1 up to 450 million cycles where the cost per operator for each setting is selected uniformly within a range of 1 million cycles that starts at the corresponding minimum cost of that setting. The results are shown in Figure 7 for the sorted broadcast. It is not a surprise that the result trends match those of Figure 5, since the impact of increasing the processing cost resembles that of increasing the ratio between the processing to tuning power consumption.

4.2.2 Scalability test

Our second experiment consists of two parts. In the first part we measure the average energy consumption per wireless client as we increase the number of queries from 20 to 300. Since we assume that each node registers one query, the number of nodes increases

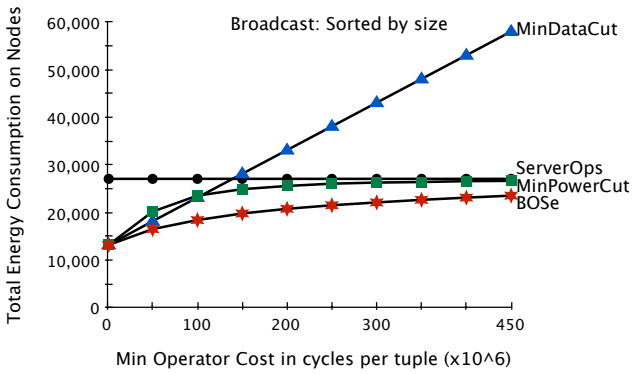

Figure 7: Total Energy Vs. the cost of each operator in cycles per tuple for sorted broadcast. The x-axis depicts the minimum number of cycles in any operator. The maximum equals the minimum plus $1 \cdot 10^6$.

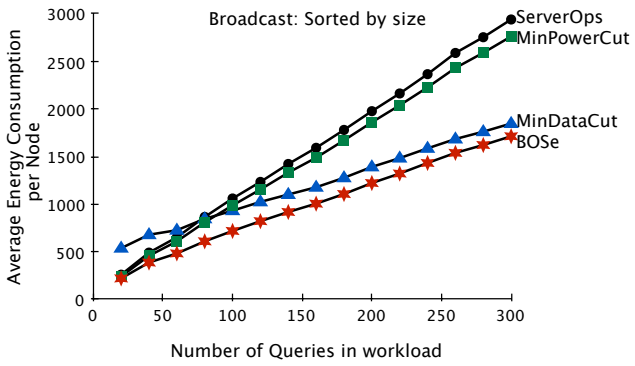

Figure 8: Average Energy per query Vs. number of queries in each workload for sorted broadcast.

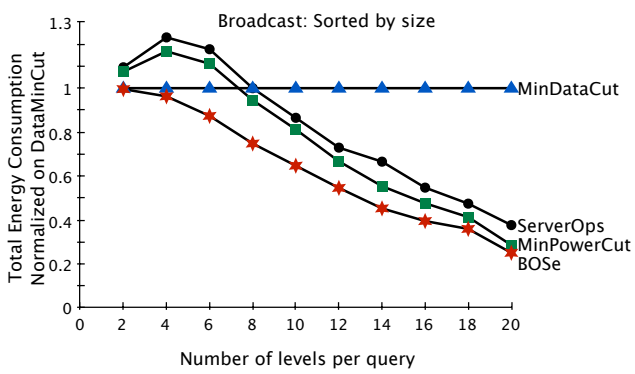

Figure 9: Total Energy Vs. number of levels per query for sorted broadcast, normalized on MinDataCut.

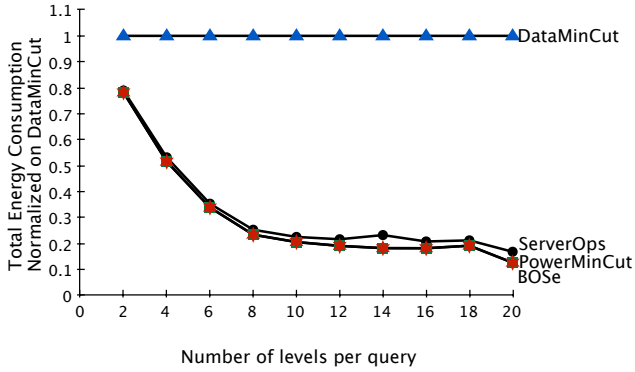

Figure 10: Total Energy Vs. number of levels per query for indexed broadcast, normalized on MinDataCut.

at the same rate as the queries. In this experiment as the number of queries increases, the size of the broadcast is also increasing. This only affects the sorted broadcast organization because every addition to the broadcast affects other clients tuning energy costs as well. In the indexed broadcast the results are the same throughout the experiment so we omitted presenting that plot. The results for the query scalability test experiment for sorted broadcast are shown in Figure 8. BOSe provides the most energy gains in both cases.

In the sorted broadcast all algorithms increase linearly, with MinDataCut and BOSe having a smaller slope. This is because both of them try to specifically minimize the broadcast size. Since this experiment only increases the broadcast size, the performance gains for both of these algorithms, as compared to ServersOps and MinPowerCut, increase as the broadcast size increases. BOSe is better because it also optimizes in terms of operator energy cost. In this experiment the improvement of BOSe over ServerOps is 42% when the number of queries is 300.

The second part of this experiment keeps the number of queries the same but increases the number of levels per query (i.e. number of operators per query) from 2 up to 20. The results are shown in Figure 9 and 10, and are normalized on MinDataCut to make them easier to understand. When the sorted broadcast is used MinDataCut scales up to 8 levels before it gets outperformed by the other algorithms. This shows that at that point the processing component outweighs the tuning component. In the case of the indexed broadcast (Figure 10), the other algorithms outperform MinDataCut from the start because the tuning component is very low compared to the processing component, thus just trying to optimize the tuning part is not enough.

5. RELATED WORK

Current DSMSs' prototypes assume that the underlying network layer is responsible for propagating the output data streams to end-users. However, this decoupling of the system from the transport layer eliminates the chance of exploiting the CQs' characteristics for better bandwidth utilization. Previous research on Publish/Subscribe and mobile information systems shows the importance of considering queries' semantics together with employing advanced data dissemination schemes such as *data multicast* and *data broadcast* (e.g., [8, 6, 4, 10, 11, 15]). In these schemes, data of interest for multiple clients is only disseminated once, thus making an effective use of the available bandwidth and allowing maximum scalability. For example, the work in [10] introduced techniques for reducing data dissemination costs in a subscription environment, by exploring the idea of merging queries with overlapping answers. The same concept can be applied in disseminating a DSMS's output data streams. That is, when multiple clients register the same CQ, the output of that query is broadcast only once. Additionally, results from overlapping CQ's can be efficiently merged to reduce

the bandwidth consumption as we previously proposed in [21, 22]. In the context of this paper, we utilize techniques for data organization that have been investigated in broadcast push, in particular broadcast indexing. Several broadcast indexing techniques have been proposed (e.g., [13]) to support selective tuning that reduces energy consumption.

The idea of query operator shipping to reduce the network utilization in a distributed database system was extensively used in MOCHA [19]. MOCHA pushes data reducing operators towards the data sources, and the data producing operators towards the clients. This is similar to our approach but in our case the data is disseminated to the clients through a broadcast network instead of point-to-point. Also, [19] considers ad-hoc queries, whereas in our work we consider CQs over data streams.

The idea of query operator distribution was proposed in distributed DSMS (D-DSMS) (e.g., Borealis [3]) for workload balancing. As the query plan and the workload may change over time, a D-DSMS needs to have the ability to dynamically move query operators across the physical machines to balance the load with the goal of improving the overall performance. There are several other approaches of CQ operator distribution in D-DSMSs that consider data transmission overhead (e.g, [25, 7]). For example, [25] aim to minimize communication cost while maintaining balance of load among hosts. Also, [18] proposes an overlay network between a D-DSMS and the physical network in order to optimize latency and network utilization based on a multi-dimensional cost space. The work in [7] incorporates knowledge of network characteristics such as bandwidth and topology into operator placement algorithms.

All of the above D-DSMS operator placement schemes only consider point-to-point unicast connection between the distributed nodes, as opposed to our work where we consider broadcast push as the means of communication. Moreover, these techniques do not take into account any energy consumption. In the future we plan to combine the goals of load-balancing, response time and energy consumption into one optimization goal.

6. CONCLUSIONS

We have developed three algorithms for continuous query operator placement for a data stream management system which broadcasts the results of continuous queries to a set of wireless clients. The goal of the algorithms is to minimize the overall energy consumption on the wireless clients. We have demonstrated the behavior of the proposed algorithms with extensive experimentation and shown that our BOSe algorithm always performs better than all the others with improvements of up to 53%.

We are currently working on extending our algorithms to support sharing of operators among multiple queries and sharing of queries among multiple clients. Also we are currently working on extending our algorithms to take into consideration energy consumption as well as response time.

7. REFERENCES

[1] Swarup A. and S. Muthukrishnan. Scheduling on-demand broadcasts: new metrics and algorithms. In *MobiCom*, 1998.

[2] D. J. Abadi et al. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[3] Daniel J Abadi et al. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[4] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *SIGMOD*, 1995.

[5] S. Acharya, M. Franklin, and S.Zdonik. Balancing push and pull for data broadcast. In *SIGMOD*, 1997.

[6] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *MobiCom*, 1998.

[7] Yanif Ahmad and Ugur Cetintemel. Network-aware query processing for stream-based applications. In *VLDB*, 2004.

[8] Demet Aksoy and Michael Franklin. R $\times$ W: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas. Operator scheduling in data stream systems. *VLDB J.*, 13(4):333–353, 2004.

[10] A. Crespo, O. Buyukkokten, and H. G. Molina. Efficient query subscription processing in a multicast environment. In *ICDE*, 2000.

[11] A. Crespo, O. Buyukkokten, and H. G. Molina. Query merging: Improving query subscription processing in a multicast environment. *IEEE Trans. Knowl. Data Eng*, 15(1):174–191, 2003.

[12] Q. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *ICDE*, 2000.

[13] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In *SIGMOD*, 1994.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MOBICOM*, 2000.

[15] W. Li, W. Zhang, V. Liberatore, V. Penkrot, J. Beaver, M. A. Sharaf, S. Roychowdhury, P. K. Chrysanthis, and K. Pruhs. An optimized multicast-based data dissemination middleware. In *ICDE*, 2003.

[16] K. Matsumura, K. Usui, K. Kai, and K. Ishikawa. Location-aware data broadcasting: an application for digital mobile broadcasting in Japan. In *MULTIMEDIA*, 2003.

[17] S. Parkvall, E. Englund, M. Lundevall, and J. Torsner. Evolving 3G mobile systems: broadband and broadcast services in WCDMA. *IEEE Communications Mag.*, 44(2):30–36, 2006.

[18] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *ICDE*, 2006.

[19] M. Rodriguez-Martinez and N. Roussopoulos. Mocha: A self-extensible database middleware system for distributed data sources. In *SIGMOD*, 2000.

[20] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Tina: A scheme for temporal coherencyaware in network aggregation. In *MobiDE*, 2003.

[21] M. A. Sharaf and P. K. Chrysanthis. Semantic-based delivery of olap summary tables in wireless environments. In *CIKM*, 2002.

[22] M. A. Sharaf and P. K. Chrysanthis. On-demand data broadcasting for mobile decision making. *MONET*, 9(6):703–714, 2004.

[23] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs. Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Trans. Database Syst.*, 33(1), 2008.

[24] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *VLDB*, 1997.

[25] Y. Zhou, B. Chin Ooi, and K.-L. Tan. Dynamic load management for distributed continuous query systems. In *ICDE*, 2005.