Optimizing the Energy Consumption of Continuous Query Processing with Mobile Clients

Panayiotis Neophytou*, Jesse Szwedko*, Mohamed A. Sharaf†, Panos K. Chrysanthis*, Alexandros Labrinidis*
*Department of Computer Science, University of Pittsburgh
†School of Information Technology and Electrical Engineering, The University of Queensland
Email: {panickos, jjs86}@cs.pitt.edu, m.sharaf@uq.edu.au, {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 system executing continuous queries (CQs), registered by the users. In this paper, we consider the situation where the results of the CQs, which are in the form of individual data streams, are disseminated to the users' hand-held, battery-operated devices over a shared broadcast medium. In order to reduce the overall energy consumption of the mobile devices, we propose BOSe*, a power-aware query operator placement algorithm that determines which part of a CQ plan should be executed at the data stream management system and which part should be executed at the mobile device. BOSe*'s effectiveness in reducing energy consumption, as well as response time under specific conditions, is evaluated using simulation, driven by parameters measured on real mobile devices.**

I. Introduction

Research in energy conservation in mobile and wireless networks has always been driven by the fact that there is a gap between the energy consumed processing data compared to the energy consumed for transmitting or receiving data at a mobile device, in favor of local processing. This is especially true as the CPUs of the mobile devices become even faster, with newer devices even featuring two cores. This fact has led to the design principle of trading off communication energy consumption for computation energy consumption (e.g., [13]). In this paper, we apply this principle to minimize the energy consumption on mobile users' hand-held, battery-operated devices linked to Data Stream Management Systems (DSMSs).

Specifically, we consider monitoring applications where mobile users register *continuous queries* (CQ), specifying events of interest to them, to be efficiently executed by a DSMS over unbounded data streams. The results of CQs are also in the form of continuous data streams that are continuously disseminated to the mobile end-users over a shared broadcast wireless medium. As an example of a CQ submitted by a mobile user, consider the query of a trader which monitors stock price updates, at the floor of a market exchange (Fig. 1): it selects the stocks that are in the NASDAQ index (operator O_1); projects out the columns of no interest (operator O_2); joins the tuples with the user's portfolio to append the buying price (operator O_3); and finally, calculates the user's profit in the last 5 minutes, every 30 seconds (operator O_4).

[1]This work was supported by NSF grants IIS-0534531 and IIS-1050301.

Our approach is to split the load of query processing between the DSMS and the mobile user devices themselves by opportunistically taking advantage of the reduced size of intermediate results to shorten the mobile devices' listening time of the broadcast and, consequently, reduce their communication energy cost, at the cost of additional local processing. In our trader query example, the first two operators O_1 (select) and O_2 (project) are data reducing, while the operator O_3 (join) and O_4 (aggregation) could potentially be data expanding. Thus, O_3 and O_4 could be shipped to the mobile client, and the much smaller intermediate result produced by O_2 will be broadcast. Further, CQs often share prefixes which include the same operators (as in Fig. 1(b)), making it even more beneficial to broadcast intermediate results shared by multiple queries since the overall broadcast size will be reduced.

In our prior work [11], we have laid the groundwork for our approach by proposing three operator placement algorithms assuming a simplistic CQ execution model. Of these, the *BOSe* (Broadcast-aware Operator Selection) algorithm was the best. Encouraged by our preliminary results, in this paper we present and evaluate *BOSe** which is based on a realistic CQ execution model that supports operator sharing among queries and sharing of queries among users.

Contributions: Our contributions are summarized as follows:

1) We propose *BOSe**, an operator placement algorithm, which takes into account the broadcast organization, sharing of operators and sharing of queries to provide the most overall energy savings at the mobile devices.
2) We present a heuristic parameterized variant of *BOSe**, namely, *BOSe-vlook*, which significantly reduces the cost for generating high-quality operator placement plans which are comparable to those of the basic *BOSe**.
3) We provide an extensive experimental evaluation, using parameters obtained from experiments using real mobile devices. Our results show that *BOSe** can achieve an overall energy reduction of up to 53% and even reduces response times.

Outline: Section II provides the system model. Section III presents *BOSe** and Section IV describes its evaluation. Section V surveys related work. Section VI lists our findings.

II. System Model

In our system, the DSMS contains a wireless disseminator module to broadcast CQ results to the mobile users (clients).

A. Data Stream Processing

A CQ evaluation plan generated by the query optimizer can be conceptualized as a data flow tree [1], [6], where the nodes are operators that process tuples and the edges represent the flow of tuples from one operator, e.g., O_x, to another, O_y (Fig. 1(a)), where $upstream(O_y) = O_x$ 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 *source* operator, e.g., O_1^k and a single *output* operator, e.g., O_4^k. Further, in a query plan Q_k, an *operator segment* $G_{x,y}^k$ is the sequence of operators that starts at O_x^k and ends at O_y^k. If the last operator on $G_{x,y}^k$ is the output operator, then we simply denote that operator segment as G_x^k.

When two or more continuous queries *share* a common sub-expression, that sharing translates into identical prefix operator segments across the plans for those queries. In such case, the DSMS continuous query optimizer typically chooses to instantiate this prefix only once. The results produced by this prefix segment are shared among the remaining suffixes of the queries that share this prefix. For each operator O_x in the shared segment, we define $\mathbb{Q}(O_x)$ as the set of queries sharing O_x. For example, in Fig. 1.(b), $\mathbb{Q}(O_1) = \mathbb{Q}(O_2) = \{Q_k, Q_j\}$. Another popular form of sharing which can be seen as a special case of operator sharing is when the same query is shared (submitted by) two or more clients. In such a case, for each shared query Q_k, we define $\mathbb{C}(Q_k)$ as the set of clients sharing Q_k. We define $\mathbb{C}(O_x)$ as the set of clients who registered queries that share O_x (e.g., $\mathbb{C}(O_x^{j,k}) = \mathbb{C}(Q_j) \cup \mathbb{C}(Q_k)$.)

In a query Q_k, an operator O_x^k (or simply O_x) could be select (σ), project (π), aggregate (e.g. $\sum$), or join-table ($\bowtie_T$). Each operator is associated with three parameters:

- c_x: the number of cycles needed to process an input tuple.
- s_x: 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.
- p_x: 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 :

- 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: $tn_x = tn_{x-1} \times s_x$
- ts_x: is the size of each tuple produced at the output of O_x after processing a tuple in its input queue: $ts_x = ts_{x-1} \times p_x$
- 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 output operator in query Q_k , then ds_x is the total size of the data block produced by Q_k.

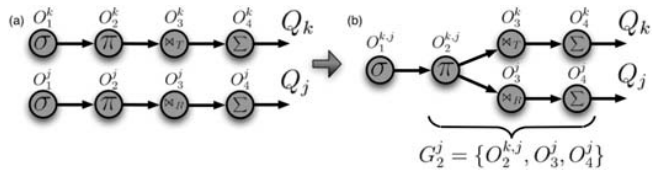


Fig. 1: Continuous Query Plan

B. Wireless Broadcast

In this work, we adopt *broadcast push* ([3]) 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. Each cycle consists of a sequence of results which could be either a final result (i.e., produced at a query's output operator) or an intermediate result (i.e., produced at a query's internal operator). In general, we denote the broadcast result produced by O_x as D_x and its size in bytes as $|D_x|$. Moreover, if O_x is shared by more than one query then those results are also shared on the broadcast and only appear once.

The result D_x of an operator O_x appears on the broadcast channel as a contiguous sequence of data packets preceded by a descriptor packet that contains an identifier of $\mathbb{Q}(O_x)$ 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 to when the client's NIC is switched to *idle mode*. Hence, the amount of energy consumed by a wireless client depends on the data organization [10], [9].

In this paper, we adopt two basic data organization schemes. The first data organization scheme uses *sorted broadcast*, where the broadcast server sorts the results according to the data size and the popularity of each result. In particular, each result D_x is assigned a priority P_x equal to $\frac{R_x}{|D_x|}$, where R_x is the number of queries that share the result D_x produced by operator O_x. That is, $R_x = \left|\bigcup_{Q_i \epsilon \mathbb{Q}(O_x)} \mathbb{C}(Q_i)\right|$. The broadcast is then organized in descending order of priority. This maintains a broadcast that follows the weighted shortest job first scheduling policy which has been shown to minimize total response time for shared resources [7].

Accordingly, a client N_i that registered a set of queries $\mathbb{Q}(N_i) = \{Q_1, Q_2, ..., Q_m\}$ will need to download a set of results (either final or intermediate) where each of those results might correspond to one or more queries in $\mathbb{Q}(N_i)$. In particular, N_i will download results $\{D_1, D_2, ..., D_n\}$ where n is the number of results and $n \leq m$. Hence, if the queries in $\mathbb{Q}(N_i)$ do not share any operators then $n = m$, otherwise $n < m$. Finally, N_i's tuning time $T_T(N_i)$ is computed as:

$$T_T(N_i) = \frac{|\,D_{min}\,| + \sum_j |\,D_j\,|}{BW}, \forall y : P_j > P_{min} \quad (1)$$

where D_{min} is the result with the minimum priority among all the results $\{D_1, D_2, ..., D_n\}$. In particular, the client will tune from the beginning of the broadcast, downloading results for queries in its registered set, and it will stay tuned until it downloads the last one (with D_{min}) in that set.

The second data organization scheme uses an *indexed broadcast*, where the broadcast server attaches an index at the beginning of each broadcast cycle (e.g., a (1,1) index [10]). The index contains an entry for each result D_x on the broadcast in the form $< \mathbb{Q}(D_x), t_x >$, where t_x is the time offset of D_x within the broadcast cycle. In this scheme, a client N_i needs to first tune to the index packet to learn the broadcast times for each of the results corresponding to the queries that it has registered for, then power off its NIC until the time of the smallest of those timestamps, say t_s. At time t_s, N_i powers on its NIC again, tunes into the broadcast to retrieve that result (i.e., D_s) and after it finishes fetching all of D_s's data packets, powers off the NIC again until the next timestamp or the next broadcast cycle, if there are no other timestamps remaining. The tuning time for a node N_i in the indexed broadcast is computed as:

$$T_T(N_i) = \frac{\sum_{x=1}^{n} |\,D_x\,| + |Index|}{BW} \quad (2)$$

where n is the number of results corresponding to N_i's queries as defined before, $|\,D_x\,|$ is the size of each of those results, and $|Index|$ is the size of the index.

C. Mobile Clients

Mobile clients, serviced by the system, can register multiple queries and then listen to a broadcast medium to get their results. We assume that each mobile client is associated with a profile that includes the following four characteristics:

- $Speed(N_i)$: processing speed of the client in cycles per unit of time.
- $P_P(N_i)$: power consumed per unit of time of processing.
- $P_T(N_i)$: power consumed per unit of time of tuning (i.e., when the NIC is active).
- $E_{PowerUp}$: energy needed to power up the NIC.

Based on the client profiles, the energy consumption and the computational cost can be computed. Specifically the tuning energy, E_{Tune} for a client N_i is as follows:

$$E_{Tune}(N_i) = T_T(N_i) \times P_T(N_i) + U(N_i) \times E_{PowerUp} \quad (3)$$

where $T_T(N_i)$ is the tuning time and $U(N_i)$ is the number of times the client needs to power up the NIC.

The processing power, $E_{Process}$ for a client N_i, given the processing time, T_P is then:

$$E_{Process}(N_i) = T_P \times P_P(N_i) \quad (4)$$

III. BOSe*: Broadcast Aware Operator Selection

In this section, we formalize the placement of CQ operators in DSMS environments with mobile clients and propose our new operator placement algorithms, *BOSe** and its variant.

A. 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 mobile clients. The total energy consumption is the sum of two components: *tuning* and *processing*, which can be expressed as:

$$E_{Total} = E_{Tune} + E_{Process} \quad (5)$$

Given the clients' profiles and their corresponding registered queries, an operator placement algorithm decides to shift some of the computation to the client if it is beneficial in reducing the overall total energy consumption. Specifically, for each query, it splits its query plan into two segments: the first segment is processed on the DSMS, whereas the second one on the clients who registered to the query. For instance, if client N_i registered queries $\mathbb{Q}(N_i) = \{Q_j, Q_k\}$ and their plans were split at operators O^j_x and O^k_y respectively, then the operators in $\mathbb{G} = G^j_{x+1,output}$ and $G^k_{y+1,output}$ have to be processed on the client. The set of operators executed at N_i is expressed as: $\mathbb{G} = G^j_{x+1,output} \cup G^k_{y+1,output}$, to cover the case where the two segments share common operators and need to be instantiated only once on each client.

Thus, for a client N_i running segments $\mathbb{G} = G^j_{x+1,output} \cup G^k_{y+1,output} \cup ...$, the processing time is computed as:

$$T_P(N_i) = \sum_{O_k \in \mathbb{G}} \frac{c_k \times tn_{k-1}}{Speed(N_i)} \quad (6)$$

Using Eq. 4 we can now find the processing energy $E_{Process}$ consumed by each wireless client N_i for use in calculating the total processing power on each client in Eq. 5

In our prior work [11], we proposed the following two algorithms along with BOSe (the preliminary version of *BOSe**.)

a) DataMinCut: minimizes the tuning energy expended by the clients. It does so by using the Max-flow Min-cut theorem, on the whole query network, which is cast into a flow-graph, to select the edges $\mathbb{E}$ whose data will populate the broadcast. Each edge in the flow graph is labeled with the average data size flowing through it.

b) PowerMinCut: attempts to minimize the overall energy by choosing the edges that result in smallest total energy (tuning and processing) for the client. It augments the edge labels with the client processing energy cost of all the operators downstream of each edge. This algorithm neglects the broadcast organization and thus may result in suboptimal energy consumption as, depending on the broadcasting scheme, a local decision may negatively affect other clients.

B. Broadcast-aware Operator Selection (BOSe*)

*BOSe** could be perceived as a hybrid of DataMinCut and PowerMinCut as it integrates the desirable features of each. On one hand, like DataMinCut, BOSe* tries to minimize the length of the broadcast cycle to minimize tuning energy. On the other hard, *BOSe**, like PowerMinCut, considers the extra energy needed for operator processing at the client.

*BOSe** uses the DataMinCut output as a starting point and then applies a greedy selection process geared towards finding a segment of operators downstream from the current cut and reinstating them back on the server. Since DataMinCut gives the minimal broadcast size, that means that any reinstatement by *BOSe** will incur an increase in the broadcast no matter what. However, 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 also depends on the broadcast organization.

At each iteration, *BOSe** examines all the current broadcast edges, $\mathbb{E}$. For each edge $E_x \in \mathbb{E}$, it generates a list of all the possible segments of operators following that edge, vertically as well as horizontally. That is, all the prefixes of the operator segments $G^j_{x+1,output}$, for all of the queries Q_j in $\mathbb{Q}(O_x)$. It does this recursively by adding to the current segment, all possible combinations of operators connected to it (power-set), taken from the next level of operators. This process is performed for each edge in $\mathbb{E}$ and the segment with the 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.

BOSe optimization function:* Recall that at each step, *BOSe** is expected to increase the tuning energy while decreasing the processing energy as compared to DataMinCut. Assume these changes are Δ_{Tune} and $\Delta_{Process}$, respectively. Hence, after BOSe* makes a selection, the overall energy consumption from Eq. (5) 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: $\Delta_{Tune} - \Delta_{Process}$ which should also be less than zero. 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 $E_R \in \mathbb{E}$. Further, assume that G is one of those prefixes under examination, which is considered as a candidate to be moved from client N_i back to the server side. For instance, G, in Fig. 2. Moving G back to the server will reduce the processing energy by the amount $\Delta_{Process}$, computed as follows:

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

Further, moving segment G back to the server entails replacing, in the broadcast $\mathbb{E}$, its input edge (i.e., edge E_R) with G's output edges, say E_{A_1} and E_{A_2} (Fig. 3). The impact of this replacement operation, on the tuning energy (Δ_{Tune}) depends on the broadcast organization.

In the case of *sorted broadcast*, removing E_R will reduce the tuning time for all clients N_i receiving data from E_R. Additionally, it will also reduce the tuning time for all the clients waiting for results that appear after E_R on the broadcast cycle ($\mathbb{N}_{\mathbb{R}}$). This reduction per client is simply ds_R/BW,

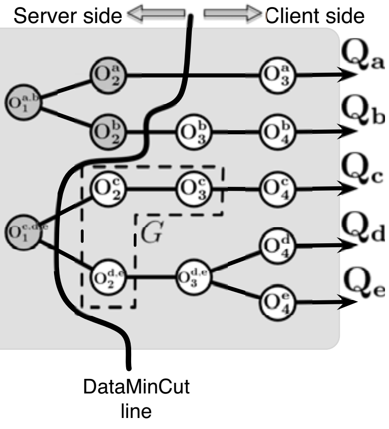
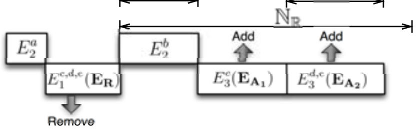


Fig. 2: DataMinCut and operator segment $\mathbb{G} = \{G^{d,e}_{2,2}, G^c_{2,3}\}$.



$\mathbb{E}_{\mathbb{R},\mathbb{A}_1}$
$\mathbb{N}_{\mathbb{A}_1}$
$\mathbb{N}_{\mathbb{R}}$
Add
Add
E_2^a
E_2^b
$E_1^{c,d,e}(\mathbf{E_R})$
$E_3^c(\mathbf{E_{A_1}})$
$E_3^{d,e}(\mathbf{E_{A_2}})$
Remove

Fig. 3: Optimization step for BOSe algorithm.

where ds_R is the data size of associated with edge E_R. Hence, the total reduction is computed as:

$$\Delta_R = \frac{ds_R}{BW} \times \left(\sum_{N_i \in \mathbb{C}(E_R)} P_T(N_i) + \sum_{N \in \mathbb{N}_{\mathbb{R}}} P_T(N) \right) \tag{7}$$

Similarly, adding E_{A_i} will increase the tuning time for the clients N_i receiving data from E_{A_i} and will also increase the tuning time for all the clients waiting for results that appear after E_{A_i} on the broadcast cycle (Fig. 3), including any newly added edges which will appear after E_{A_i}. This increase per client is simply ds_{A_i}/BW, where ds_{A_i} is the data size associated with edge E_{A_i}. This increase is computed as:

$$\Delta_A = \sum_{A_i \in \mathbb{E}_{\mathbb{A}}} \left(\frac{ds_{A_i}}{BW} \times \left(\sum_{N \in \mathbb{C}(E_{A_i})} P_T(N) + \sum_{N \in \mathbb{N}_{\mathbb{A}_i}} P_T(N) \right) \right) \tag{8}$$

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

Moreover, under a sorted broadcast, the location of the new edge E_{A_i} on the broadcast cycle is determined according to each one's data size. Since $ds_{A_i} > ds_R, \forall i$, then E_{A_i} will appear at a further offset than E_R's one. Therefore, when E_{A_i} replaces E_R, the clients N_i which consume results in E_{A_i} will have to spend more tuning time to receive them than what they used to spend to receive E_R. This translates into extra tuning energy which is computed as:

$$\Delta_{R,A} = \sum_{A_i \in \mathbb{E}_{\mathbb{A}}} \sum_{N \in \mathbb{C}(E_{A_i})} \frac{\sum_{E \in \mathbb{E}_{\mathbb{R},\mathbb{A}_i}} ds_E}{BW} \times P_T(N) \tag{9}$$

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

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 energy of the client under examination without any impact on the other clients in the system (i.e., $\Delta_{R,A} = 0$ in Eq. 9). Thus, Δ_{Tune} for the indexed broadcast is denoted by $\Delta'_{Tune} = -\Delta'_R + \Delta'_A$, where equations 7 and 8 become:

$$\Delta'_R = \sum_{N \in \mathbb{C}(E_R)} \frac{ds_R}{BW} \times P_T(N) \tag{10}$$

$$\Delta'_A = \sum_{A_i \in \mathbb{E}_{\mathbb{A}}} \sum_{N \in \mathbb{C}(E_{A_i})} \frac{ds_A}{BW} \times P_T(N) \tag{11}$$

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

Complexity and Approximation: The BOSe* algorithm is performing an exhaustive search over all possible combinations of operators both vertically and horizontally. The number of combinations for each edge in $\mathbb{E}$ is $\sum_i^{log_d n} 2^i$. To reduce the search space we propose a parameterized version of BOSe, called *BOSe-vlook(x)*, which limits the cardinality of the set taken from the power-set (as mentioned before) up to x.

IV. Experimental Evaluation

In this section, we illustrate the performance of our algorithms using our own simulator driven by parameters determined from experimental results using a mobile device.

A. Experimental Setup

The algorithms – DataMinCut, PowerMinCut, and *BOSe** and BOSe-vlook(x) – are implemented in Java and run as they would on a real system. As a base case, we also implemented ServerOps which executes all the operators on the server and broadcasts the final results to the clients.

The workload used in the simulator is a complete plan of the registered CQs, along with the mobile clients' profiles which include their speed and power consumption parameters for processing and tuning.

We measure the total energy consumption at all the mobile clients as the ratio of processing to tuning power of the nodes increases linearly from 0.01 to 0.65, while keeping the tuning power constant. The remaining parameters are summarized in Table I. It should be noted that from our measurements on real hardware we found these ratios to be between 0.0104 and 0.6572. The values reported are averages of 15 runs.

B. Experimental Results (Energy)

Sorted Broadcast: As Fig. 4 shows, *BOSe** outperforms all other algorithms. ServerOps' performance is constant because it runs all operators at the server, and thus increasing the processing power of mobile clients will have no impact on its performance. It also shows that DataMinCut is linearly increasing because DataMinCut tries to minimize only the tuning energy and not the processing energy. Hence, DataMinCut selects the same set of edges for every setting regardless of the increase in power consumption for processing. PowerMinCut performs better than DataMinCut and ServerOps; however, it 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. *BOSe** always performs the best because it evaluates the different options considering both the broadcast organization and the processing power costs of operators running on the mobile clients, thus striking a fine balance between tuning and processing energies. For instance, at a processing to tuning energy ratio of 0.01, *BOSe** provides an improvement in energy of 50% over ServerOps and at 0.35 an improvement of 14%. In the extreme case of 0.65, it still provides a small improvement.

The heuristics *BOSe-vlook(2)* and *BOSe-vlook(3)* perform very similarly to *BOSe**. The reduction in number of computations is very significant: 38% for *BOSe-vlook(3)* and 75% for *BOSe-vlook(2)*, while the performance remains very close to the best (up to 99%) for *BOSe-vlook(3).*

Indexed Broadcast: Due to space limitations we omit these results; however they are similar to those seen in Fig. 4. For example, the energy savings in this broadcasting scheme are 53% compared to ServerOps, at a ratio of 0.01. In this case *BOSe** and PowerMinCut perform identically as the broadcast

TABLE I: Workload Default Characteristics

Levels per query refers to the number of operators which exist in every single-stream query in the workload. Operator cost skewness is a 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 $0.2 \times i$.

Parameter	Values
Number of queries	50
Number of clients	25
Number of clients per query	1-5 (Zipf)
Levels per query	10
Sharing levels	2
Maximum Degree of Operator Sharing	3
Sources tuple rate	500 – 1000 tuples/sec uniform
Sources tuple size	2000 – 4000 bytes uniform
Selectivity	0.2 – 1.8, uniform
Projectivity	0.5 – 1.5, uniform
Operator costs (cycles/tuple)	$100 \times 10^6 - 200 \times 10^6$
Operator cost skewness	0.2 increments (Zipf)
Hand-held device speed	1×10^9 cycles/sec
Server speed	1×3.4^9 cycles/sec
Bandwidth	125000 bytes/sec

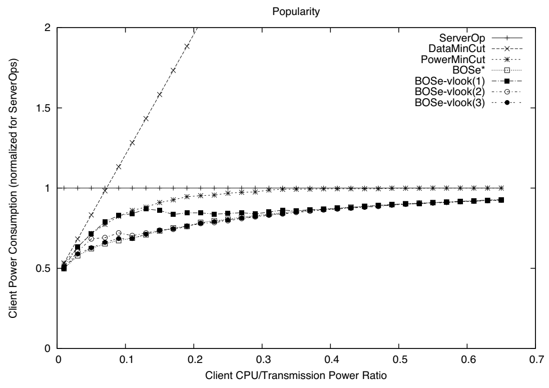


Fig. 4: Energy consumption for Tuning Vs Processing power cost for sorted broadcast, normalized over ServerOps.

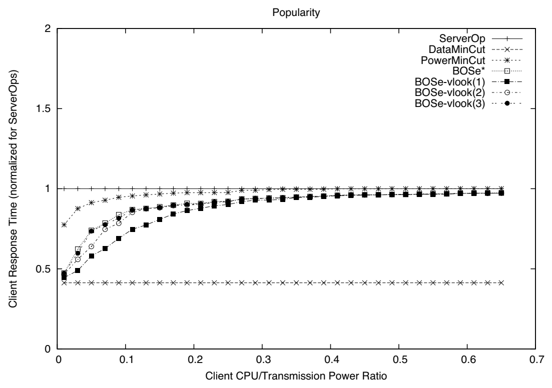


Fig. 5: Average response time of client queries for Tuning Vs Processing power cost for sorted broadcast, normalized over ServerOps.

order does not affect the power consumption of the clients.

C. Experimental Results (Response Time)

Although we were optimizing for power consumption on the client, we noticed notable gains for the response time of client queries (Fig. 5). This can be attributed to the fact that while the server handles all queries (we assume a round-robin scheduler) the client need only be concerned with processing operators for queries that it is interested in and can dedicate all its resources to that end. Thus, by broadcasting an intermediate result, the client can complete the computations quicker. However, this is only true when the following equation holds for a client N_i (without loss of generality assume only one query for N_i):

$$T_{T_{ServerOps}} + T_P(Server) \geq T_{T_{BOSe}} + T_P(N_i) + T'_P(Server)$$

If the workload remains the same, we know that $T_P(Server) \geq T'_P(Server)$ and $T_{T_{ServerOps}} \geq T_{T_{BOSe}}$ where T'_P is the new server processing time under *BOSe**. This means that $T_P(N_i)$ (the time to process on client N_i) has to be low enough to not dominate the other components. In general, longer processing time on a client means higher energy drain so *BOSe**, while minimizing power consumption, will also reduce response time, since long-running operators will not be moved to the client. However, clients with efficient CPUs may see a higher response time as BOSe* may choose to move more operators to these clients, thus increasing response time while decreasing power consumption.

V. Related Work

Current DSMSs' 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 (e.g., [5], [4], [10]). 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. In this paper, we apply the same concept in disseminating a DSMS's output data streams for scalability and energy savings.

The idea of query operator shipping to reduce the network cost in a distributed database system was used in MOCHA [12] where data reducing operators were pushed 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, [12] 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., [2]) for workload balancing. Similar to our approach, several approaches of CQ operator distribution in D-DSMSs consider data transmission overhead (e.g., [15], [14], [8]). All the D-DSMS approaches consider point-to-point unicast connections between the distributed nodes, as opposed to our work that considers broadcast push. Further, these techniques do not consider energy consumption.

VI. Conclusions

In this paper, we proposed *BOSe**, a query operator placement algorithm that splits the load of CQ processing between the DSMS and the mobile user devices in order to minimize the overall energy consumption on the mobile devices. Our extensive experimental evaluation showed the effectiveness of *BOSe** in distributing the query operators in a way that reduces the size of the query results that need to be continuously disseminated to the mobile users, BOSe* leads to an overall reduction up to 53% in energy in mobile devices, compared to the traditional centralized CQ processing at the DSMS.

References

[1] D. J. Abadi et al. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.
[2] D. J Abadi et al. The design of the borealis stream processing engine. In *CIDR*, 2005.
[3] S. Acharya, M. Franklin, and S.Zdonik. Balancing push and pull for data broadcast. In *SIGMOD*, 1997.
[4] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *MobiCom*, 1998.
[5] D. Aksoy, M. Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.
[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas. Operator scheduling in data stream systems. *VLDB J.*, 13(4):333–353, 2004.
[7] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4), 2007.
[8] P. Pietzuch et al. Network-aware operator placement for stream-processing systems. In *ICDE*, 2006.
[9] Q. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *ICDE*, 2000.
[10] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In *SIGMOD*, 1994.
[11] P. Neophytou, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis. Power-aware operator placement and broadcasting of continuous query results. In *MobiDE*, 2010.
[12] M. Rodriguez-Martinez, N. Roussopoulos. MOCHA: A self-extensible database middleware system for distributed data sources. In *SIGMOD*, 2000.
[13] M. A. Sharaf and P. K. Chrysanthis. On-demand data broadcasting for mobile decision making. *MONET*, 9(6):703–714, 2004.
[14] Y.Ahmad and U. Cetintemel. Network-aware query processing for stream-based applications. In *VLDB*, 2004.
[15] Y. Zhou, B. Chin Ooi, and K.-L. Tan. Dynamic load management for distributed continuous query systems. In *ICDE*, 2005.