2011 12th IEEE International Conference on Mobile Data Management

Visualization of Energy Consumption of Continuous Query Processing with Mobile Clients

Jesse Szwedko*, Panayiotis Neophytou*, Panos K. Chrysanthis*, Alexandros Labrinidis*, Mohamed A. Sharaf†
*Department of Computer Science, University of Pittsburgh
†School of Information Technology and Electrical Engineering, The University of Queensland
Email: {jjs86, panickos, panos, labrinid}@cs.pitt.edu, m.sharaf@uq.edu.au

***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, event detection is carried out by a central server executing continuous queries. In this demonstration, we focus on the case where users with mobile devices submit continuous queries (for event detection) to a data stream management server which disseminates the results to the users over a shared broadcast medium. In order to minimize the overall energy consumption of the mobile devices (clients), we have proposed operator placement algorithms that split the processing of each continuous query between the centralized server and the requesting mobile clients, thus trading off energy consumption for communication energy consumption for computation. Specifically, in this demonstration, we present an interactive graphical interface to the inner workings of our three proposed operator placement algorithms, whereby attendees are able to investigate various query plans and the decisions that the algorithms make, as well as visualize the results of these algorithms in terms of client power consumption and response time. Besides being able to step through an algorithm's execution as it considers various operator placement decisions, attendees are able to experiment with different scenarios by customizing the parameters of the query workloads (e.g., changing the selectivities and projectivities of the operators) or the client's profile (e.g., power consumed per unit of time of processing) and examine the impact.**

I. Introduction

Complex event detection over unbounded 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. Examples of such applications, of equal interest to stationary and mobile users, include: monitoring of stock quotes, airline schedule updates, local news, weather readings, traffic information and emergency management. Typically, event detection is carried out by a *data stream management server* (DSMS) executing *continuous queries* (CQ), that have been previously registered by the users of the monitoring applications [1], [2].

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. First, it filters out stocks that are not in the NASDAQ index using the *select* operator. Then, it drops columns of no interest (such as source information etc.) using the *project* operator. Then, it joins the tuples with the user's portfolio to append the buying price using the *join* operator. Finally, it calculates the user's profit in the last 5 minutes, every 30 seconds, using the *aggregate* operator.

In applications where end-users are mobile, users usually submit CQs from hand-held battery-operated devices and receive the results of CQs over a shared broadcast wireless medium. The results of CQs are also in the form of continuous data streams that need to be continuously disseminated to the mobile end-users. Our goal in [3], [4] was to design operator placement algorithms that work together with the broadcast organization to minimize the total energy consumption on the hand-held devices. The approach was based on our experimental observations that the energy costs for receiving data are typically much higher than the energy costs for processing said data, an observation that has also motivated in-network processing in wireless sensor networks (e.g., [5], [6]). This is especially true as the CPUs of these devices become even faster, with newer devices even featuring two cores.

The operator placement algorithms we proposed choose to split the load of query processing between the DSMS and the mobile user devices (mobile clients) themselves by opportunistically taking advantage of the queries which demonstrate fluctuation in the size of intermediate results, between their operators. This means that the system will broadcast the smaller intermediate results of some queries, thus keeping the broadcast smaller, and leave the rest of the processing to the clients registered to those queries, in essence trading some of the energy previously utilized for communication for increased computational energy costs.

This paper summarizes our three proposed operator placement algorithms to provide context for the demonstration and then outlines the graphical user interface (GUI) that we have developed to display how these algorithms work and how well they perform. This interactive GUI allows the user to load individual system workloads, view the query plan and broadcast network organization, specify the computation and communication power properties of individual mobile devices, and easily discover the decisions the various algorithms made on where to place the operators. The user may also modify the query plan itself by modifying parameters of the query operators or add/removing operators and clients.

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

978-0-7695-4436-6/11 $26.00 © 2011 IEEE
DOI 10.1109/MDM.2011.70

Contributions: In summary, in this demonstration proposal we present a GUI front-end for:

1) Visualization of the query network state and the choices the various algorithms make at each step of their execution.
2) Visualization and comparison of the results of the operator placement algorithms in terms of client power consumption and access time.
3) Investigation of the effects of various system parameters on results.

Outline: In Section II, we describe the parameters of the system model. Section III, provides an overview of our three proposed operator placement algorithm. Lastly, in Section IV, we describe the graphical user interface we developed as well as the equipment and demonstration settings, and how the attendees would interact with our system.

II. System Model

As described in [4], we assume a realistic system model where a DSMS allows mobile clients to register and share multiple CQs. In addition to the standard modules of admission manager, query optimizer, scheduler, memory manager and load shedder, the DSMS implements a wireless disseminator module which broadcasts the results of CQs to the mobile clients. In order to allow for extensive experimentation over a variety of variables, we have developed a detailed simulator that considers all the key characteristics of our system model which we describe below.

A. Data Stream Processing

For the data stream processing model, we identify a number of parameters for both the operators in the query plan as well as the clients. A CQ evaluation plan generated by the query optimizer can be conceptualized as a data flow tree [2], [7], where the nodes are operators that process tuples and the edges represent the flow of tuples from one operator to another. This operator tree is converted into a vertex operator tree where operators with shared input edges are placed in the same vertex. The operator placement algorithms then decide which edges to cut upon and move the right hand side of the cut to the clients. In a query, each operator could be one of four types: select (σ), project (π), aggregate (e.g., $\sum$), or join-table ($\bowtie_T$) and is associated with the following parameters:

1) The number of cycles needed to process an input tuple.
2) The number of output tuples produced after processing one input tuple (selectivity). This is less than or equal to 1 for a filter operator and it could be greater than 1 for a join operator.
3) The size of a tuple produced by the operator compared to its size before being processed (projectivity). This is less than or equal to 1 for a project operator and it may be greater than 1 for a join operator.

B. Wireless Broadcast

We adopt broadcast push as the mechanism to disseminate query results to clients, since it naturally complies with the DSMS access model where a client installs a CQ once and the server repeatedly transmits 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.

The amount of energy consumed by a wireless client depends on the data organization [8], [9], [10]. The two possible broadcast organizations we consider are:

- *Sorted:* The DSMS sorts the results according to the data size and the popularity of each result and broadcasts in that order. Each client must listen to the broadcase from the beginning until it receives its results. This weighted shortest job first scheduling policy has been shown to minimize total response time in shared resources [11].
- *Indexed:* The DSMS attaches an index at the beginning of each broadcast cycle. The client must listen to the entire index and then subsequently may only tune in and listen to its particular result. This selective tuning enabled by an indexed broadcast has been shown to minimize energy consumption at the expense of response time [10].

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). The energy cost of receiving the results depends on the tuning power consumption of the client (as described below) and the datasize of the broadcast.

C. Mobile Clients

Mobile clients, serviced by the system, can register multiple queries and then listen to a broadcast medium to get their results. The parameters associated with the client are:

1) The processing speed of the client in cycles per unit of time.
2) The power consumed per unit of time of processing.
3) The power consumed per unit of time of tuning (i.e., when the network interface card (NIC) is active).
4) The energy needed to power up the NIC.

Based on the client profiles, the energy consumption and the computational cost can be computed for use by the algorithms. Specifically, for each client N_i, the tuning energy is computed as:

$$E_{Tune}(N_i) = T_T(N_i) \times P_T(N_i) + U(N_i) \times E_{PowerUp} \quad (1)$$

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, and the power consumed per unit time processing, $P_P(N_i)$ is then:

$$E_{Process}(N_i) = T_P \times P_P(N_i) \quad (2)$$

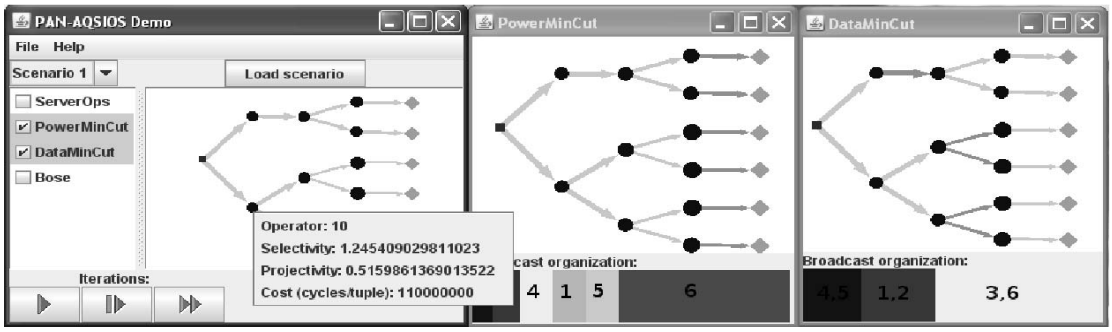

Fig. 1: GUI Interface for investigating the execution of the BOSe* algorithm and visualizing the results. The leftmost window comprises the initial application. From here the user can load scenarios, modify the system parameters through a point-and-click interface on the query plan, and advance the BOSe* algorithm. The next window shows the results of the PowerMinCut algorithm and the final shows the results of the DataMinCut algorithm.

Additionally the response time is computed as:

$$R_T = T_{P_{Server}} + A_T + T_{P_{Client}} \quad (3)$$

where $T_{P_{Server}}$ is the amount of time between the data arrival at the DSMS until the data is available to the disseminator. A_T is the *access time*, i.e., the time it takes a client to receive the data from the broadcast, and $T_{P_{Client}}$ is the time it takes a client to process the data on the locally-placed query operators. As discussed in the introduction, our approach to minimizing the overall power consumption (shown in Eq.4) is based on the observation that the energy costs for receiving data are typically much higher than the energy costs for processing that data.

III. Operator Placement Algorithms

We include a short description of each of the algorithms included in the demonstration. The goal of each of these algorithms is to optimize the following equation and are described in more depth in the cited works.

$$E_{Total} = E_{Tune} + E_{Process} \quad (4)$$

a) DataMinCut[3]: minimizes the tuning energy expended by the clients by labeling every edge with the average data size flowing through and discovering the MinCut of the graph. Broadcasting the data flowing through the MinCut of the graph guarantees that the broadcast size is minimal.

b) PowerMinCut[3]: 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 the edges that minimize for one client may cause other clients to expend additional energy as they appear later in the broadcast.

c) BOSe (Broadcast aware Operator Selection)[4]:* tries to minimize the overall energy consumption of clients similar to PowerMinCut. The difference is that BOSe* considers the effect on the broadcast organization when deciding which edges to broadcast. It begins by considering the graph cut that DataMinCut would choose. It then applies a greedy selection process to find a segment of operators within each query (or combination of queries sharing operators) and reinstate them back on the server. Since the cut it starts with 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. As such, BOSe* considers the global effect as reinstating operators may change the broadcast organization and adversely effect other clients.

IV. Demonstration Scenario

For the demonstration, we have developed a graphical front-end that allows attendees to step through the BOSe* algorithm, the best performing of our three algorithms, and visually evaluate and compare its results with those of DataMinCut, PowerMinCut, and ServerOps in various scenarios. ServerOps is the base case against which the others algorithms are compared. It executes all operators on the server and transmits the results to listening clients. This is the way that current DSMSes operate for mobile clients. Attendees may tweak the parameters of the system (as described below) to their liking to see the resulting effect on the algorithms.

A. Front-end

The front-end is a Java-based GUI that allows the user to visualize the execution and results of the system. It begins by allowing the user to choose from a list of predefined scenarios to view. Each of these scenarios loads a particular workload (which describes all of the parameters of the system) into the simulator. The query plan is graphically presented to the user as seen in Figure 1.

This query plan graph provides the user with a simple point-and-click interface for discovering information and editing parameters. The relative computational costs of the operators are shown by varying the size of the ellipse representing the operator (i.e., larger ellipses correspond to more computationally intensive operators). Similarly, edges between operators that carry larger data sizes are shown to be relatively larger.

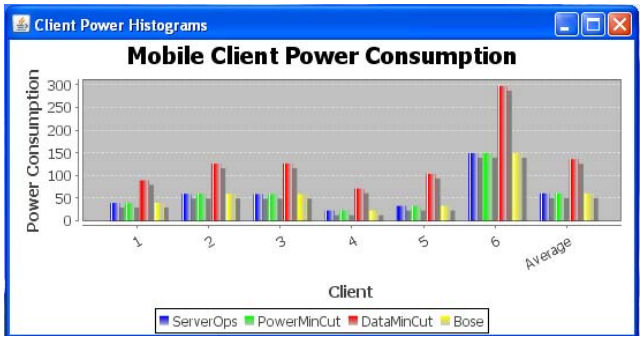

Fig. 2: Histogram of the client power consumption results.

Each of the segments of the graph is clickable and provides the following information when clicked:

- *Source Operator:* The input rate and size of each tuple.
- *Operator:* The selectivity and projectivity of the operator.
- *Edge:* The tuple rate flowing along the edge and the total datasize.
- *Client:* The cost to the client for each query it is listening to under the naive ServerOps algorithm.

Right-clicking will allow the user to edit each of the parameters for operators. Right-clicking the client will allow the user to tweak the various components of the client profile such as the computational power of the client. After editing, the user may rerun the simulation using the new parameters. The user may also delete operators or clients by double clicking and choosing to delete. The user may drag between two nodes to add new edges between them. If the user drags from a node into whitespace they will be prompted to add an operator or client at that location.

Clicking on the name of any of the algorithms in the list on the left brings up a separate frame showing the same query graph as ServerOps, but with the results for that given algorithm including the cuts along which the algorithm chooses to broadcast (the remaining operators then being executed on the clients after intermediate data transmission) and the cost to each of the clients; this is done by coloring the edges along which the graph is cut. Users may compare these results side-by-side. Below the query graph is a visual representation of the broadcast plan at any given cycle. The segments are broadcast in order from left to right (and are annotated with which queries are represented by that segment). Clicking a segment displays additional information about that segment.

In order to visualize the execution of the BOSe* algorithm we withhold execution of the algorithm initially and provide controls for the user to iterate through the steps of the algorithm. At each step, the graph is updated to display the current subset of operators that BOSe* is considering pulling back to the server along with its calculation of the benefit. Because BOSe* considers many states, a Fast Forward button is also provided to quickly view the iterations of the algorithm.

Finally, the user may view the overall results of the algorithm across all clients in the Histogram Pane (Figure 2).

B. Settings

For the conference demo we will use several laptops to run the application that will be preloaded with a number of scenarios that will showcase various parameters of the system. Attendees will be able to use the system to execute various workloads, tweak system parameters, and view the results of the algorithms in a graphical manner for easy comparisons. We will also have a mobile device on hand to demonstrate how we collected the statistics used to determine the appropriate power consumption rates that we use in our simulations.

C. Implementation

The GUI interface and simulator were written using Java 1.6. The GUI relies on the Javax Swing API as well as the Prefuse library [12] for the query plan and results visualization. The interface interacts with the standalone algorithm implementation module by feeding it a configuration file based on the scenario chosen. The interface then collects the statistics regarding the given scenario and uses this information to populate the interface with the query plan, graph cuts, and power results. Editing the parameters in the GUI is made possible by manipulating the internal data structures of the simulator and then rerunning the algorithms. The interactive histogram of the client power consumptions was made possible through JFreeChart [13].

The simulator takes, as input, a workload file specified by the given scenario which it uses to instantiate the system model. The simulator uses the Mascopt library ([14]) internally to represent the query network and run the various optimization algorithms and derive that statistics.

REFERENCES

[1] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Algorithms and metrics for processing multiple heterogeneous continuous queries," *ACM Trans. Database Syst.*, vol. 33, no. 1, 2008.
[2] D. J. Abadi *et al.*, "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.
[3] P. Neophytou, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Power-aware operator placement and broadcasting of continuous query results," in *MobiDE 2010*, June 2010.
[4] P. Neophytou, J. Szwedko, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Optimizing the energy consumption of continuous query processing with mobile clients," in *MDM*, 2011.
[5] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," in *OSDI*, 2002.
[6] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *VLDB J.*, vol. 13, no. 4, pp. 384–403, 2004.
[7] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas, "Operator scheduling in data stream systems," *VLDB J.*, vol. 13, no. 4, pp. 333–353, 2004.
[8] D. Aksoy and M. J. Franklin, "Scheduling for large-scale on-demand data broadcasting," in *INFOCOM*, 1998.
[9] A. Crespo, O. Buyukkokten, and H. G. Molina, "Efficient query subscription processing in a multicast environment," in *ICDE*, 2000.
[10] Q. Hu, W.-C. Lee, and D. L. Lee, "Power conservative multi-attribute queries on data broadcast," in *ICDE*, 2000.
[11] N. Bansal and K. Dhamdhere, "Minimizing weighted flow time," *ACM Trans. Algorithms*, vol. 3, no. 4, 2007.
[12] J. Heer, S. Card, and J. Landay, "Prefuse: a toolkit for interactive information visualization," in *SIGCHI*, 2005.
[13] D. Gilbert, "The jfreechart class library," *Object Refinery Limited*, 2008.
[14] J. Lalande, M. Syska, and Y. Verhoeven, "Mascopt-a network optimization library: Graph manipulation," 2004.