Hybrid Push-Pull Query Processing for Sensor Networks

Niki Trigoni, Yong Yao, Alan Demers, Johannes Gehrke
Cornell University
{niki,yao,ademers,johannes}.cs.cornell.edu

Rajmohan Rajaraman
Northeastern University
rraj@ccs.neu.edu

Abstract: A powerful database abstraction for sensor networks has recently emerged in which clients program the sensors using a *declarative* query language. Existing work assumes that data is either pushed from sensor nodes to a gateway, or data is pulled from the gateway through queries. We show that hybrid *push-pull* data dissemination outperforms the pure approaches and offers significant energy savings.

1 Introduction

Sensor networks consisting of small sensor nodes with sensing, computation and communication capabilities will become ubiquitous. Recently, a database approach to programming sensor networks has gained interest [BGS00, YG03, MFHH02]: Clients "program" the sensors through queries in a high-level language (such as variants of SQL), and catalog management and query processing techniques abstract the user from the physical details of tasking the sensors. We call the resulting system a sensor data management system (SDMS). Sensor networks have important constraints on communication, computation and power consumption. Energy is the most valuable resource for unattended battery-powered nodes. Since radio communication consumes most of the available node power, SDMSs apply different strategies to minimize communication such as in-network processing.

There are two ways of executing queries. The first is to reactively send queries into the network and to *pull* relevant results out of the network. Another possibility is to proactively *push* all possibly relevant readings out of the network independent of the current queries. In this paper we propose a hybrid *pull-push* approach: sensor readings are pushed proactively to selected nodes in the network from where they are later pulled when queries are asked. We claim that carefully drawing the line between the pull and the push areas can offer significant communication savings. Our approach is orthogonal to data-centric storage [RKY$^+$02], as we store readings at the push-pull boundary for query processing purposes, and it is complementary to energy-efficient MAC and topology control schemes. Most related is work on materialized view selection in database systems [ACN00].

Our model is the following. Instead of considering a single query we consider multiple long-running queries with a probabilistic query and sensor update workload. The key contributions are as follows: i) for a probabilistic sensor update scenarios, we show that a hybrid pull-push approach is often preferred to the pure approaches; ii) given a tree that connects all nodes to the gateway, we propose an adaptive distributed algorithm that selects an optimal hybrid configuration; iii) in a thorough experimental evaluation, we quantify the benefits of our technique. To our knowledge, this work is the first to study the pull-push model in the context of sensor networks.

2 Problem space

We divide time into *rounds*. Queries are executed at the end of a round, and they refer to sensor readings generated during that round. Let $QW = \{(Q_1, p_1), \ldots, (Q_n, p_n)\}$ be the query workload, where p_i is the probability that Q_i is executed at the end of a round. Similarly, let $SW = \{(s_1, u_1), \ldots, (s_k, u_k)\}$ represent the sensor update workload, where u_i is the probability that sensor s_i is updated during a round. We assume that both query and sensor update probabilities are independent and discuss the implication of removing the independence assumption in section 3. For concreteness, we consider queries Q_i of the form: $select\ sum(s.attr)\ from\ Sensors\ s\ where\ s.loc\ in\ Region_i$. We are given a communication tree of sensors rooted at the special node called the gateway. The energy cost of sending an n-bit message along a tree edge is $\alpha + \beta n$, where α is the startup cost of activating an edge and β represents the per-bit cost. In-network processing along tree routes is suitable for a number of environmental, monitoring and surveillance applications.

Let $QW = \{(a + b, 1 - \epsilon), (a + c, 1 - \epsilon)\}, SW = \{(a, 1), (b, 1), (c, 1), (i, 1), (r, 1)\}$ be the traffic workload in the tree of figure 1 (in QW we use a node identifier to denote its value). We show that the optimal pull-push configuration depends on the expected query and result costs at each edge. Assume a simple energy model $\alpha + \beta n$ where $\alpha = 0$ and $\beta = 1$. Let q be the expected cost of sending a query request message down a tree edge, and $r >= q$ the cost of sending a data result message. The expected *pull* cost of edge $r \leftrightarrow i$ is $q + 2(1 - \epsilon)r$, representing an unconditional request message and two query results with independent probabilities $(1 - \epsilon)$. The expected *push* cost of this edge is $2r$. Edges $i \leftrightarrow b$ and $i \leftrightarrow c$ have equal expected *pull* costs of $q + (1 - \epsilon)r$. Edge $i \leftrightarrow a$ has expected *pull* cost $q + (1 - \epsilon^2)r$. The expected $push$ cost of edges near the leaves is r.

The optimal hybrid solution is now completely determined by the relative values of q, r and ϵ. When $q > 2\epsilon r$, a completely proactive (push) solution is best. For $\epsilon r < q < 2\epsilon r$, the best solution is to make on-demand (pull) only the edge $r \leftrightarrow i$, proactively sending data from the leaves to node i. For $\epsilon^2 r < q < \epsilon r$ it becomes beneficial to make the edges $i \leftrightarrow b$ and $i \leftrightarrow c$ on-demand as well, but still materialize the value of a at node i. For $q < \epsilon^2 r$, a completely on-demand solution is optimal. This example illustrates that query probabilities affect our choice of where to draw the line between the push and pull edges. Similar examples can be given to show the effect of sensor update probabilities.

Pull-push decisions were made based on the expected costs of query and result messages at different edges. In the general case, multi-query optimization techniques complicate the task of computing these costs. Let $QW = \{(a + b, 1), (a + b + c, 1), (c, 1)\}, SW = \{(a, 1), (b, 1), (c, 1), (i, 1), (r, 1)\}$ be the traffic workload in the tree of figure 1. Since queries are deterministic a "push" strategy is preferred. Interior nodes of the tree compute sub-aggregates of the values they receive from their children, and forward them up the tree towards the root. Multi-query optimization involves recognizing when the values of sub-aggregates can be shared effectively among queries, so that redundant data messages can be eliminated. The three queries $a + b$, $a + b + c$, and c are not linearly independent –the values of any two of them can be used to calculate the value of the third. Thus, node i should forward only *two* of the values (say $a + b$ and c) to the root r. The root may then compute the third value ($a + b + c$) locally, achieving a net saving of energy.

This technique of "reducing" the set of data values forwarded toward the root can be repeated bottom-up at every subtree. Queries in QW are first projected to a subtree rooted at node N which contains, say, ℓ sensor descendants. The projection is represented as a $n \times \ell$ bit matrix M, where $M(i,j)$ is 1 if query Q_i accesses sensor s_j. M is reduced to echelon form. The row cardinality (rank) of the reduced matrix M' denotes the number of results that node N must send to its parent. Under deterministic query and sensor update workloads the number of results that every node forwards to its parent is easy to evaluate (as the rank of the corresponding projected query matrix). However, when queries are probabilistic, the expected rank of the projected query matrix is very hard to evaluate. Things only become more complex when sensor updates are probabilistic. Enumerating all different combinations of queries and sensors that might occur in a round and evaluating the rank of the corresponding matrices is prohibitive, especially for sensor nodes with limited processing capabilities. The expected traffic routed over an edge is critical in deciding whether to apply the pull or push model; with the example above, we showed that the expected traffic may be hard to compute analytically.

3 An adaptive hybrid pull-push approach

Our algorithm works in two phases and selects an optimal pull-push strategy that addresses two issues: i) depending on the aggregate function and the multi-query optimization techniques applied, the expected volume of local edge traffic could be hard to compute; ii) by synthesizing locally-optimal decisions based on the expected traffic, we can obtain an incompatible pull-push configuration (e.g. select a pull edge below a push edge).

Simulation phase: This is a statistics gathering phase. Nodes monitor the traffic of the network for a certain number of rounds (say m rounds). At every round, each node keeps record of the query q and result traffic r routed through it using the pull model. It also calculates the size of results R that it would forward to its parent, had it *not* known the current queries, i.e. had it used the push model. At the end of m rounds, every node evaluates the local average sizes $avg(q)$, $avg(r)$ and $avg(R)$ of the forwarded query and result messages.

Dynamic Programming (DP) phase: By the end of the simulation phase, every node N has evaluated the average cost of applying the push or pull model at the local edge $e_{N \leftrightarrow P(N)}$: $Push_N = \alpha + \beta * avg(R)$ and $Pull_N = 2 \times \alpha + \beta * (avg(q) + avg(r))$. The optimal compatible pull-push configuration is selected in two passes. In the bottom-up pass, every node N recursively evaluates and informs its parent about the following costs:

$STPull_N = Pull_N + \sum_{i=1}^{childNo_N} STPullPush_{ch_N[i]}$

$STPush_N = Push_N + \sum_{i=1}^{childNo_N} STPush_{ch_N[i]}$

$STPullPush_N = min\{STPull_N, STPush_N\}$

In the top-down pass, a node N (except for the root) waits until it is informed about the *model* (pull or push) used by its parent. Initially *model=pull* for every node. The *model* value of the current node is set to *push* if either $model_{P(N)} = push$ or $STPush_N < STPull_N$. Node N then broadcasts its own $model$ value to its children. By the end of the top-down phase, an optimal pull-push model has been assigned.

Algorithm evaluation: We use the standard technique of Monte Carlo simulations to ob-

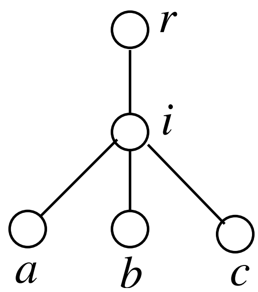

Figure 1: A tree example

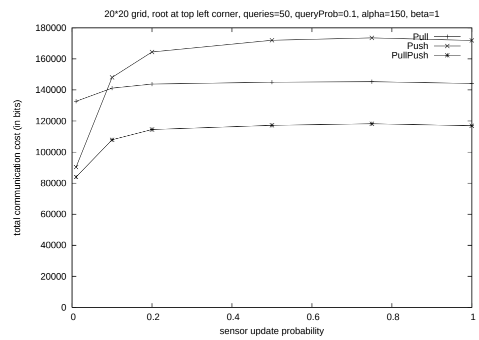

Figure 2: Impact of update probabilities

tain near-accurate estimates of the required query and result costs. Let $Y(U_1, \ldots, U_k)$ be a function of independent query or update events U_i. Y may stand for either q, r or R cost. Let $\overline{Y}$ denote $\sum_j Y_j/m$ of m samples from the underlying probability space. If $Y_{\max}$ is the maximum possible value for Y, then for any $0 < \varepsilon < 1$, $\Pr[|\overline{Y} - E[Y]| > \varepsilon] \leq e^{-\varepsilon^2 m/Y_{\max}}$. The proof is omitted for lack of space. By setting the number of samples m sufficiently larger than $Y_{\max}$, we can set the probability that the estimate is ε-away from the expectation arbitrarily close to zero. The independence assumption for query and update probabilities is used in order to bound the error between the estimated and the expected values. The proposed adaptive algorithm would be applicable even without the assumption, but without providing optimality guarantees. The DP phase enforces compatibility constraints for a hybrid model, i.e. an edge can be assigned the pull model, iff all ancestor edges are also pull. It can be implemented in a distributed manner and communication-wise it involves sending two small messages per node. The calculations that it involves are very simple and do not require large storage capabilities. In order to adjust to changes in traffic probability distributions, the adaptive algorithm is repeated periodically (at a frequency that depends on the network dynamics).

4 Experiments

We simulate a network of 400 nodes organized in rectangular grid. A tree connects all nodes to the gateway (located in the bottom left corner). In every round, we run multiple sum queries that cover all sensors in a rectangular area. The area dimensions are randomly chosen between 1 and 20. Query messages are bit-vectors denoting which queries in the probabilistic workload QW occured at the current round. Result messages include a bit-vector, which denotes which sensors in the subtree are updated in the current round, and a set of reduced query results. Each query result has size 32 bits.

We first study the performance of different models on a workload of 50 queries with small probabilities (0.1). Figure 2 shows that the push model outperforms the pull method for low update probabilities. The hybrid pull-push model outperforms the other models in all cases offering benefits of up to 20%. Figure 3 shows the impact of query probabilities on different models, when the update probability is low (0.1). For query probability close to

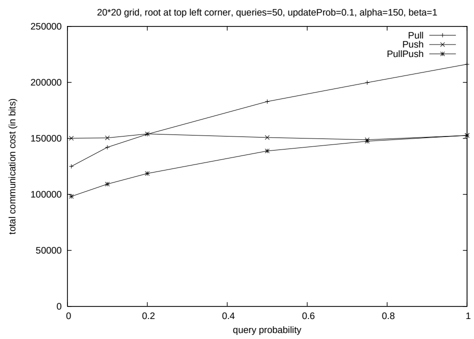

Figure 3: Impact of query probabilities

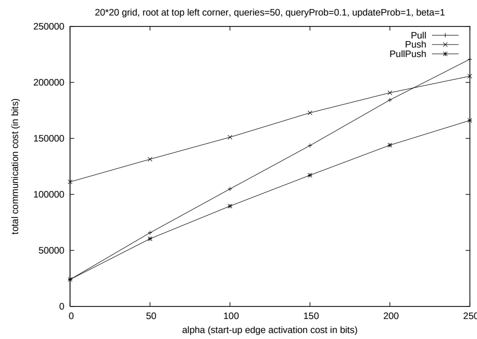

Figure 4: Impact of edge start-up cost

0.2, the pull and the push cost become equal and the relative benefit of the hybrid approach (25%) is maximized. We finally evaluate the role of the edge activation cost α for small query probabilities (0.1) and deterministic updates (figure 4). When α is very small, the pull outperforms the push model since the overhead of query requests is small compared to their filtering benefits. Push is preferred however for large αs. The relative benefit of the hybrid model is maximized when the pure model costs are equal ($\alpha = 210$). This point shifts to the left, if we increase query or decrease update probabilities.

5 Conclusions and Future Work

Our work proposes a hybrid pull-push paradigm for data dissemination in sensor networks. experiments show that a hybrid paradigm offers significant benefits in a variety of scenarios. Measurements were done in the context of multi-query optimization techniques that intrinsically complicate pull-push decisions. The optimal pull-push strategy is identified by an adaptive and simple distributed algorithm suitable for sensor networks. In the future, we plan to study the benefits of a hybrid strategy for correlated traffic patterns. We would also like to study the role of the tree structure in making pull-push decisions.

References

[ACN00] Agrawal, S., Chaudhuri, S., und Narasayya, V. R.: Automated selection of materialized views and indexes for SQL databases. In: *Proceedings of 26th International Conference on Very Large Data Bases*. S. 496–505. 2000.

[BGS00] Bonnet, P., Gehrke, J., und Seshadri, P.: Querying the physical world. *IEEE Personal Communications*. 7(5):10–15. 10 2000.

[MFHH02] Madden, S. R., Franklin, M. J., Hellerstein, J. M., und Hong, W.: Tag: A tiny aggregation service for ad-hoc sensor networks. In: *OSDI*. 2002.

[RKY+02] Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., und Shenker, S.: Ght: A geographic hash table for data-centric storage. In: *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*. 2002.

[YG03] Yao, Y. und Gehrke, J.: Query processing in sensor networks. In: *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*. Asilomar, California. January 2003.