An Integration Framework for Sensor Networks and Data Stream Management Systems

Daniel J. Abadi MIT · Wolfgang Lindner MIT · Samuel Madden MIT · Jörg Schuler Tufts University

Computer Science and Artificial Intelligence Lab.
Massachusetts Institute of Technology
{dna, wolfgang, madden}@csail.mit.edu

Dept. of Computer Science
Tufts University
js@cs.tufts.edu

Abstract

This demonstration shows an integrated query processing environment where users can seamlessly query both a data stream management system and a sensor network with one query expression. By integrating the two query processing systems, the optimization goals of the sensor network (primarily power) and server network (primarily latency and quality) can be unified into one quality of service metric. The demo shows various steps of the unified optimization process for a sample query where the effects of each step that the optimizer takes can be directly viewed using a quality of service monitor. Our demo includes sensors deployed in the demo area in a tiny mockup of a factory application.

1 Introduction

Collections of tiny, radio-equipped, battery-powered sensing devices, or *sensornets* promise to collect a significant amount of interesting data about the world around us. Unlike conventional remote (satellite-based) sensing which typically consists of large images, sound files, or scientific data processed by individually developed software, much of the data from sensornets promises to be small, tuple-oriented, and structured. Typical applications include sensors that continuously report the temperature of refrigerated boxes being shipped across the country to detect spoilage or theft, or sensors that report the light and temperature values in various offices in a building as input to an HVAC (Heating, Ventilation, and Air Conditioning) control system. In recent years we have seen the development of sensor network query-processing systems (SNQPs), most notably Cougar [1] and TinyDB [8].

Alone, these systems provide a convenient interface for extracting data from sensornets. Furthermore, query optimization appears to be a promising approach for reducing sensornet power consumption [11, 7, 4], the primary sensor performance metric [6, 10]. Despite the promising advances seen in the research, we believe there is a substantial opportunity to further improve both the power efficiency and utility of SNQPs by their integration with one of several streaming database systems, such as STREAM [9], TelegraphCQ [5], or Aurora [3], commonly referred to as data stream management systems (DSMSs).

This integration offers three major benefits:

- The ability to combine stored or streaming data from the DSMS with data from the sensornet. For example, users may want to compute a join to decide if readings from a motion-detector are correlated with network activity, or if truck locations match the expected locations on the planned route.
- A single, integrated interface for interacting with both the streaming database and the sensor network. TinyDB currently allows users to log query results into a RDBMS table via JDBC, but queries must still be input to TinyDB independently of the RDBMS. By providing a single, seamless system, users are only required to learn to configure and interact with one set of interfaces.
- The ability to optimize between the database system and the sensor network. For example, it may be desirable to push certain filters and aggregates into the sensor network if user queries are interested in only particular subsets or coarse summaries of readings.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

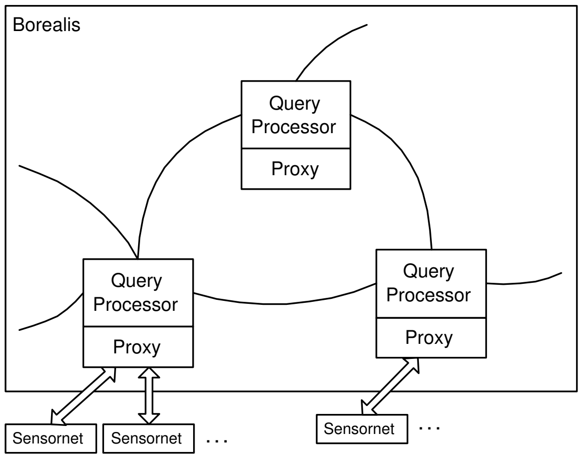

Figure 1: General Architecture

We have begun the development of a next generation stream processing system, called Borealis [2], with a team from Brandeis, Brown, and MIT. This demonstration shows one piece of this larger project where the stream processing engine seamlessly queries a SNQP system.

Our demo includes sensors deployed in the demo area in a tiny mockup of a factory application; users can query both Borealis and TinyDB, and see the power consumption and efficiency of execution of their queries. A detailed description of our demonstration is provided in Section 4. Next we describe the general architecture of our system with a description of a few of the technical challenges.

2 Architecture

The architecture allows the sensor network to communicate its processing capabilities and constraints, along with descriptions of the data it can produce to the DSMS. It also provides a translation mechanism so that query operations entered through the DSMS application interface can be performed in the sensornet should the system choose to do so. In addition, it provides a way to resolve contrasting optimization goals of sensor networks and DSMSs. Finally, the architecture is sufficiently general so that future efforts to integrate other forms of data sources (with limited query processing capabilities) with DSMSs (such as xml-streams, web sources or even other DSMSs) can be worked into the same architecture.

Figure 1 shows an overview of our architecture. Here we show a DSMS (in our demonstration, a Borealis prototype) distributed across 3 sites, two of which receive data from sensor networks. The fundamental idea behind the integration architecture is to place a proxy intermediary at each interface between sensor networks and instances of the DSMS. The role of the proxy is to make the sensor network appear to the DSMS instance as if it were another instance running on a different site. The DSMS can then move operators and stored relations across the server/sensor boundary with the same ease as if it were moving across server/server boundaries.

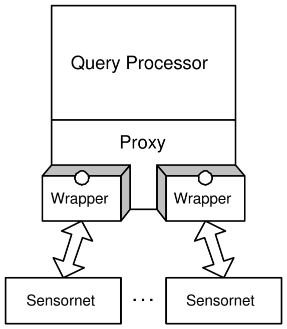

Figure 2: Proxy and Wrapper

We assume that each sensornet is uniquely connected to just one DSMS instance and that the sensornet is capable of performing some portion of the query plan. The proxy has three primary functions. First, it gathers statistics about constraints of connected subnetworks so that it can reject proposals from the DSMS to move operators or stored relations into the network (or change the parameters of operators already in the sensor network) that the sensor network does not have the processing power, storage, or bandwidth capacity to handle. Second, if it accepts an operator on behalf of the sensor network, it selects the appropriate implementation of the operator. Finally, it works with the DSMS to optimize the query (with respect to application QoS). This third function is described in Section 3. These functions are performed by the proxy for all connected sensor networks. However the proxy needs information about the current state of each connected subnetwork (e.g. how much available bandwidth is being used or how much power is remaining). For this reason, the proxy interacts with its sensornets using a *wrapper* for each network as shown in Figure 2. The wrapper provides a standardized API to integrate the connected sensornets into the optimization process. That includes a set of meta data which describes the related sensornet. All additional data sources to the DSMS are connected through these wrappers.

3 Query Optimization

Query optimization across a DSMS-sensor network integration faces three primary challenges. First, the processing capabilities of the sensor network is much smaller than the capabilities of the DSMS and thus only a subset of potential query plans (mappings of query operators to nodes upon which they will be run and implementations of these operators) can actually be executed.

Second, in contrast to a DSMS, in a sensor network the same operator will be run on multiple nodes (for standard SNQPs such as TinyDB). Thus, the decision is not where to put an operator in the sensor network, but whether to place an operator in the network at all. Such a decision must be made in a centralized manner since the result affects multiple nodes. The proxy architecture described above provides a simple solution to both of these problems in that the proxy serves as a central decision maker that is cognizant of the processing capabilities of the underlying network and can accept or reject query plans and operator movements using knowledge of these capabilities. For example, if the sensor network does not support windowed aggregations, the proxy can reject all query plans that involve the network performing a windowed aggregate operator.

The third challenge is that a DSMS and a sensor network have different and potentially conflicting goals. The DSMS is responsible for maximizing the QoS to end applications, and thus is interested in decreasing latency, increasing throughput, and maximizing the quality of results delivered to these applications. In contrast, a sensor network is primarily concerned with minimizing power consumption in order to extend lifetime and reduce cost. In some cases, these goals conflict. As a case in point consider whether or not to place a join operation (of sensor data with a static table) in a sensor network. Assume that the join predicate is highly selective, but that the static table is large so that the table must be horizontally partitioned and the join performed in parallel on many sensors. Assuming the network has the processing capability for the operator, it is in the sensor network's interest to perform the operation in-network since it reduces the number of transmitted tuples and thus power used. Due to the lossy nature of wireless communication, tuples generated by sensors might not reach nodes storing parts of the static join table, leading to lost join results. This affects the quality of results observed by the end application. While the sensor network might want to perform the join in-network, the DSMS might prefer to have the network just transmit the original data and perform the join once the data reaches the wired interface.

We attempt to solve this problem by aggregating sensor network constraints into one lifetime metric and supplementing DSMS QoS with an additional dimension to optimize. The lifetime of a query is equal to the minimum of the lifetimes of all data sources of this query. Lifetime can be thought of as a fixed amount of service the DSMS has bought from the sensor network. When the service runs out, no data will be produced for the query. This effectively terminates the query as a consequence. The more work the query requires of a sensor network, the faster the fixed amount of service is used. Thus, there is cost observable to the application in making decisions that might improve latency or quality while also increasing power utilization in the sensor network. This allows the DSMS to choose in which direction of the power/latency/quality trade-off to optimize using application QoS functions.

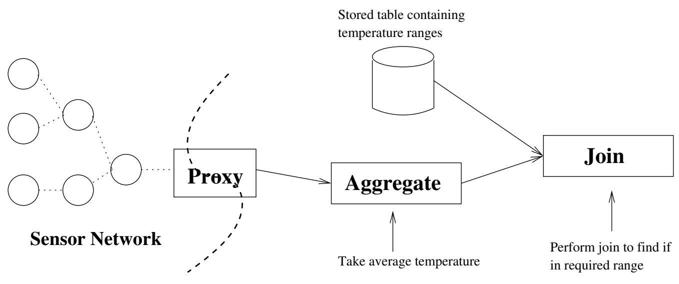
Stored table containing temperature ranges
Proxy
Aggregate
Join
Sensor Network
Take average temperature
Perform join to find if in required range

Figure 3: Query Shown as a Workflow Diagram

4 Demonstration Highlights

This demonstration shows a simulated factory environment with temperature sensitive construction phases. The factory produces a product whose creation requires the temperature to remain in a small, fixed range that varies over time. Should the temperature fall outside this range, the product is in danger of being damaged and action must be taken immediately. A continuous query is thus desired that joins aggregate temperature readings from sensors located at various positions in the factory with a time-indexed relation that encodes the desired temperature range. Should the temperature ever fall outside the required range, an appropriate reaction can be taken (such as to halt production until temperature falls within the required range).

We simulate this factory query with a small factory replica built out of Lego and placing inside it a portable air conditioner and several Mica motes that can sense ambient temperature.

The continuous query demonstrated is (in a derivative of TinySQL [8]):

```
SELECT a.atemp, a.anum
FROM schedule AS s,
     (SELECT AVG(temp) AS atemp,
             COUNT(temp) AS anum
      FROM sensors) AS a
WHERE s.ts > t.tsmin AND
      s.ts < t.tsmax AND
      a.atemp > t.tempmin AND
      a.atemp < t.tempmax
```

This query can be more clearly shown in a workflow diagram as in Figure 3.

As shown, the sensors relay their temperature readings through the proxy to the DSMS which then aggregates these readings and joins them with a stored table. There is clearly an alternative choice for where the aggregation and join can be performed: the sensor network. However, the decision of where to perform these operators has a variety of consequences on the QoS dimensions of power utilization, tuple loss rate, and latency. Further, the system has the option of changing the sensor sample rate to maximize the lifetime-tuple quality trade-off.

The demonstration shows, through a QoS monitor graphical interface, how QoS varies with system optimization decisions of operator movements and sensor sample rates. Other monitors show the results in each individual QoS dimension of these optimization decisions (the user can see the lifetime QoS increase when the aggregate operator moves into the sensor network and the quality dimension increasing at the same time as lifetime decreasing as the sensor sample rate is increased). The effects on latency, loss rate, and lifetime of moving the join operator into the sensor network depend on the size of the static table and the selectivity of the join predicates. We allow the user to send updates to the join table to observe how changing these statistics increases or decreases QoS upon moving the join into the network. We also allow the user to change the temperature of the air conditioned "factory" and to change the parameters of the query to observe the DSMS/sensor network integration successfully run the query.

5 Conclusion

It is clear that the integration of sensor networks and DSMSs is important. Sensor networks serve as a natural data source to DSMSs, and in return DSMSs are capable of executing much more complex operations on the data than nodes in the network, allowing a wider variety of queries to be performed on sensor produced data. Integration of these systems to create a unified query plan that will execute across DSMS/sensor boundaries is not a trivial task because of the different architectures and assumptions of these systems. Sensor networks (using SNQPs such as TinyDB) perform the same operation in parallel across many different nodes in a lossy and power constrained environment. The DSMS performs only one instance of an operation on a server node with fewer power, CPU, and storage constraints. Optimization of this query plan presents further difficulties. We demonstrate a successfully integrated sensor network and DSMS where user queries can be run and optimized across these heterogeneous query processing components.

References

[1] Cougar web page. `http://www.cs.cornell.edu/database/cougar/`.

[2] Daniel Abadi, Yanif Ahmad, Hari Balakrishnan, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, John Jannotti, Wolfgang Lindner, Samuel Madden, Alexander Rasin, Michael Stonebraker, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the borealis stream processing engine. (submitted for publication).

[3] Daniel J. Abadi, Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, September 2003.

[4] Boris Bonfils and Philippe Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the First Workshop on Information Processing in Sensor Networks (IPSN)*, April 2003.

[5] S. Chandrasekaran, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. January 2003.

[6] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCOM*, Boston, MA, August 2000.

[7] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *ACM SIGMOD*, 2003.

[8] Samuel Madden, Wei Hong, Joseph M. Hellerstein, and Michael Franklin. TinyDB web page. http://telegraph.cs.berkeley.edu/tinydb.

[9] R. Motwani, J. Window, A. Arasu, B. Babcock, S.Babu, M. Data, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation and resource management in a data stream management system. In *CIDR*, 2003.

[10] Greg Pottie and William Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51 – 58, May 2000.

[11] Yong Yao and Johannes Gehrke. Query processing in sensor networks. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.