Entirely Declarative Sensor Network Systems

David Chu Arsalan Tavakoli Lucian Popa Joseph Hellerstein
EECS Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
{davidchu,arsalan,popa,hellerstein}@cs.berkeley.edu

ABSTRACT

The database and sensor network community have both recognized the utility of SQL for interfacing with sensor network systems. Recently there have been proposals to construct Internet protocols declaratively in variants of Datalog. We take these ideas to their logical extreme, and demonstrate entire distributed sensor network systems built declaratively. Our demo exposes the rapidity, flexibility, and efficiency of our approach by building several fully-functional yet widely-varying sensor network applications and services declaratively. As a result of our declarative construction, we are able to highlight a wealth of previously underexposed similarities between sensor networks and database concepts. In addition, we tackle many database systems challenges in building multiple layers of a declarative database for an embedded, distributed system.

1. INTRODUCTION

The database and sensor network community have both recognized the utility of declarative interfaces to the sensor network. TinyDB, arguably the most successful sensor network application to date, exposes the sensor network as a data stream management system with an adapted SQL interface [7]. For some applications, this approach lends familiarity, safety and sufficient flexibility.

Yet, as with embedded systems, sensor networks exhibit tight couplings with their intended applications. This often means systems designers and application developers choose systems languages, e.g. C, because high-level interfaces do not support particular features (expressiveness) or do not offer acceptable performance (efficiency). Understandably, this comes at the expense of increased development complexity, error-prone implementations, and difficulty in expressing global behavior and high-level optimizations.

This demo presents a third point in this design space. We propose the use of deductive declarative query languages as the systems language for sensor networks. In particular, we use a variant of Datalog, *SNlog*, to construct entire sensor system stacks in our distributed deductive declarative database system, *DSN*. Modeling the sensor network as a deductive declarative database makes a lot of sense since the central process of the sensor network is to manage the generation, transformation and movement of data. In sensor networks, each system stack is composed of several protocols, services and applications. We have found the following classes of protocols, services and applications particularly amenable to the declarative approach:

- *End-user applications:* Event tracking applications; passive monitoring applications.
- *Data presentation services:* SQL-like interfaces; publish-subscribe interfaces; distributed inference algorithms.
- *System management services:* In-network system reprogramming services; node localization and coordinate establishment algorithms; distributed data storage services.
- *Network protocols:* Routing protocols such as prototypical many-to-one, one-to-many and point-to-point protocols; Both datagram and stream-oriented transport protocols with varying reliability, in-order delivery, data object size properties

For some of the above classes, we demonstrate several different instances of the class in DSN. These applications then have different different storage, localization, routing, etc. needs.

The idea of casting routing and overlay protocols as deductive databases has recently been proposed in [6, 5, 4]. SNlog is derived from Overlog, the declarative query language proposed in [5]. The benefits of declarative networking transfer directly to sensor networks: expressiveness of the language, efficiency of the resulting protocols, and safety properties in line with Datalog's termination guarantees.

In addition, entirely declarative sensor networks also offer several additional benefits. First, querying for sensor data is straightforward. The range of declarative data acquisition options that made TinyDB attractive are similarly available through DSN's SNlog. Second, with the projected decreasing costs of Flash memory, data storage is an area of rapidly increasing interest. Since DSN already implements a database, DSN offers a natural solution here as well, such as cost-based interfacing with Flash backing stores. Third, DSN provides straightforward support for event detection. Our use of a logic language naturally lends itself to composing higher level events (e.g. "Car Detected") from lower

VLDB '06, September 12-15, 2006, Seoul, Korea.

level events (e.g. "Magnotemeter Sensor" and "Vibration Sensor"). Fourth, DSN provides data independence: DSN helps shield sensor network programmers from the slew of hardware platform changes. It is particularly important in the sensor network space, where the rate introduction of new platforms and sensor hardware is commensurate with the rate of new applications. Hence, application writers can concentrate on defining application logic; hardware vendors can focus on introducing new sensors and platforms. Lastly, DSN allows concise declarations of entire system behavior, and not just the routing or overlay layer. Not only do we use declarative rules to specify connections between application, service and protocol components, we also implement these components with declarative rules. The declarative rules for a given component are rarely more than a dozen lines. Entire system stacks are often less than one hundred lines.

In the following sections, we show several SNlog examples (Section 2), provide an overview of the system architecture (Section 3), and discuss the demo of DSN (Section 4).

2. SNLOG BY EXAMPLE

To give a sense for the declarative approach, we present several SNlog examples. In addition to regular Datalog conventions, at most one parameter of each tuple is marked with an "at" sign (@). This denotes that the parameter is a nodeid type and that when the parameter is fixed, the destination of the tuple is the parameter's value. We discuss how we process these rules in Section 3. For now, we focus on the structure of SNlog declarations.

2.1 Simple Collection and Dissemination

Our first example, Listing 1, is a data collection application that gathers data from all nodes in the network at the base station. Despite its simplicity, it is appropriate for many simple sensor network deployments. This prototypical sensor network application is easily expressed with a single rule and query.

Listing 1: Collection and Dissemination

```
CD1: store(@Y,Oid,Object):- produce(@X,Oid,Object),
       consume(@Y,Oid).
Query: store(@Y,Oid,Object).
```

Rule CD1 is read as: if a node X produces an object data identified as Oid, and the node Y consumes Oid, then this Object that is currently stored at X should also be stored at Y. The query sets the goal the system aims to solve; in this example, store(@Y,Oid,Object) collects every neighbor node's sensor data at Y. To collect facts at base we need to specify facts of the form 2:

Listing 2: Collection Facts

```
consume(@base,Oid).
```

The produce predicate is a builtin predicate that will generate tuples based on sensed data. If a consume predicate is true at the base and produce events get fired up at nodes, then the system will store the produced objects at base.

The same rule set of Listing 1 can support another very common sensor network service, namely data dissemination. This is typically a subprocess of network control, e.g. reprogramming the sensor network with a new executable or disseminating a new SQL query in TinyDB. For that we need to specify facts as in Listing 3. Note that in this situation the builtin predicate produce will activate at the base station and not on the nodes.

Listing 3: Dissemination Facts

```
consume(@node1,Oid1).
consume(@node2,Oid1).
consume(@node1,Oid2).
```

At the level of abstraction of the collection application, many service details (network, storage, etc.) are hidden from the user, and hence this particular application interface is similar to TinyDB. It is already apparent from this first example that although data requires movement across the network, the SNlog specification does not decide how this is accomplished. This a rich area for optimization for both the compiler and runtime. We mention one possible well-known network routing protocol in the next section.

2.2 Tree construction

In-network spanning-tree routing is a well-studied sensor network routing protocol. Coincidentally, tree construction is a special case of the Internet's Distance Vector Routing (DVR) protocol. Hence, our tree construction protocol is adapted from that of declarative DVR presented in [6]. The basic idea is that constructing an network spanning tree is equivalent to the transitive closure of shortest network paths. For the sake of space, we refer the reader there to a full discussion of this example.

2.3 Version coherency

Previous listings showed a data dissemination example. In such situations, it is often desirable to provide coherency as well, e.g. all nodes are running the same disseminated SQL query. Various sensor network eventual consistency protocols provide such version coherency , Listing 4 illustrates a declarative implementation of one such protocol [3].

Listing 4: Version coherency

```
VC1: compare(@Y,X,Oid,V1,V2) :-
       refreshEvent(@X), link(@X,Y,C).
       version(@X,Oid,V1), version(@Y,Oid,V2),
VC2: response(@Y,X,Oid,V1,V2) :-
       compare(@Y,X,Oid,V1,V2), V1<V2.
VC3: compare(@X,Y,Oid,V2,V1) :-
       response(@Y,X,Oid,V1,V2).
VC4: request(@Y,X,Oid,V1,V2) :-
       compare(@Y,X,Oid,V1,V2), V1>V2.
VC5: store(@Y,Object) :- store(@X,Object),
       request(@Y,X,Oid,V1,V2), resolve(Oid,Object).
Query: store(@X,Object).
```

In rule *VC1*, refreshEvent, a periodic, system-generated tuple, initiates a comparison between a node X's local version V1 and each neighbor Y's version V2 of item Oid. This comparison then leads to either a request by X for the new data from its up-to-date neighbor (*VC4* and *VC5*), or a response to update its out-of-date neighbor (*VC2*). The response is expressed as a recursion leading to a subsequent request (*VC3*).

Not shown are three additional rules for manipulating timers associated with refereshEvent.

3. SYSTEM ARCHITECTURE

Our architecture, shown in Figure 1, consists of four major components. The SNlog declarative specification language provides a rule-based interface for defining the system. The

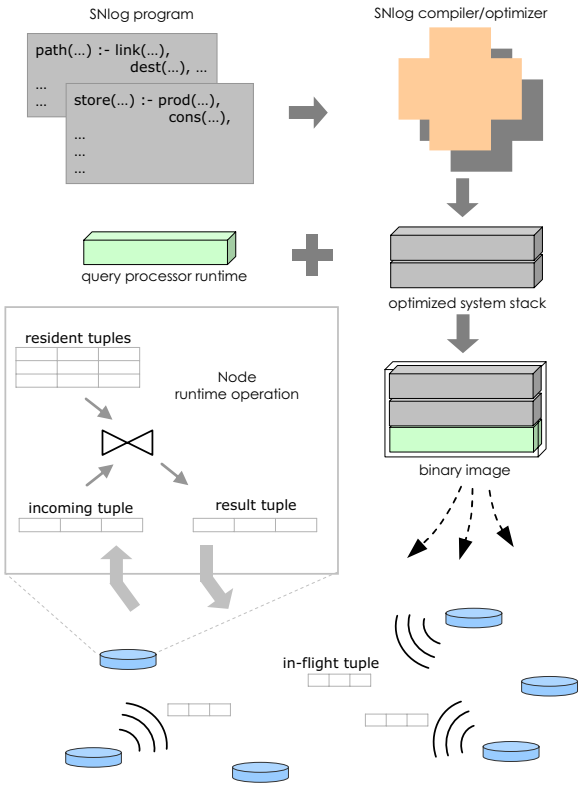

Figure 1: DSN Architecture. SNlog is compiled and distributed to the network, at which point each node executes the query processor runtime. In addition, not shown above, the runtime supports the dynamic introduction of new queries.

compiler/optimizer converts the specification into a binary image, which then is programmed into all nodes in the network. Finally, the runtime query processor on each node processes each query in order to produce the correct tuple output. We examine each component in more depth below.

3.1 SNlog

As mentioned earlier, SNlog is a modified version of Datalog that is able to specify certain physical properties such as where tuples are stored and sent. We have further modified it to provide additional features specific to sensor networks. This provides a set of rules that define everything above the link layer, in essence the network, transport, and application layer, while assuming certain basic services provided by the link layer, such as link discovery, neighbor table population, etc. Furthermore, a set of possible queries that can be posed are specified as well.

3.2 Compiler

The compiler translates the SNlog specification to create requisite on-node data structures, and translates this specification, along with the list of potential queries, to create a system stack combined with a runtime query processor for each sensor node. The constrained resources of sensor nodes make full fledged query processors infeasible. The limited query processor is acceptable as sensor networks are generally tasked with very specific applications, and hence do not need to be able to process a wide variety of queries.

The system is designed for TinyOS [2], the de facto operating system for sensor networks, which uses nesC [1], a modified version of C. Consequently, all our code is converted into nesC during the compilation process. We architect our system for use with certain builtin predicates, such as SP, a link layer abstraction for sensor networks [8]. The final step is to compile the query processor and the underlying structure into a single binary image for distribution.

3.3 Code Installation

The method for disseminating code in the form of binary images has been relatively well studied for sensor networks. We make the simplifying assumption that the initial binary image are already download onto each node prior to deployment. One desire may be to change the binary images on the nodes during runtime, perhaps to modify the query processor to handle new classes of queries. Our demonstration shows that such dissemination protocols can easily be implemented by including them as part of a declarative service specification in the system stack.

3.4 Runtime Query Processor

The query processor on each node allows for dynamic queries to be inserted into the network. Queries are no longer restricted to data gathering and analysis. As our demonstration shows, networking functions such as collection, dissemination, topology building, and robust applications can be designed through the dissemination of queries to systems that maintain the list of rules initially specified. Runtime operations consist of nodes locally processing incoming tuples by joining with resident tuples according to the SNlog specification. Result tuples are reinserted into the set of resident tuples, potentially causing the execution of more rules. Certain rules specify that result tuples are required by other nodes in the network. In this case, these tuples are shipped to their destinations.

3.5 Challenges and Optimizations

In addition, there are several technical systems challenges that DSN addresses.

- *Significant resource constraints:* The declarative SNlog translation to an operational system stack, and the DSN runtime must operate extremely efficiently. Sensor networks are rarely granted abundant resources. This indicates that simple ports of existing deductive declarative databases are unlikely to succeed.
- *Amorphous library interfaces:* Entire system stacks invariably are compromised of many components. SNlog not only provides the ability to define new components, but also manages inter-component relationships. Performing this interfacing easily is not straightforward, yet fundamental to growing a large SNlog component library.
- *Low-level system operations:* DSN provides several built-in primitives to interface with low-level system operations. Extensibility for interfacing with new device drivers, such as those for a new type of sensor, is also crucial.

Tackling these challenges effectively is of imminent importance in any practical sensor network system. We pro-

Figure 2: Internals of the Trio mote sensorboard

vide several optimizations designed to make implementing an entire declarative networking system feasible. The majority of these deal with minimizing the number of messages transmitted and received, as the radio is the most power-consuming component for a sensor node. First of all, in cases where possible, when a node receives a query, it piggybacks the result onto any outgoing queries in the same packet. Second, when the application allows it, nodes process queries in a passive manner; instead of broadcasting a request for a certain predicate required to process a query, the node simply waits to receive it. Finally, we take advantage of the fundamental broadcast nature of the link medium to avoid the typical naive unicast communication.

4. DEMONSTRATION

Our demonstration showcases the system in its entirety, by allowing the user to interactively specify applications using SNlog, and then observe the performance of the application when it is implemented on a sensor network testbed.

4.1 Testbed

The testbed will consist of 10 sensor nodes arranged in a grid format, and equipped with a wide variety of sensors, including acoustic and passive infrared. Figure 2 shows the internals of a Trio node. Every node is connected through a USB backchannel to the base station, a laptop. The USB connection is used to upload the initial binary image onto each node. All other communication is performed using regular radio transmissions and receptions.

With a USB backchannel, each node is additionally instrumented to a real-time debugger, deployed in parallel with the actual application. For a given running of the system, we are able to step through the execution of SNlog on nodes. Every node periodically sends its state, as well as any packets that it receives, over the USB to the base station to create a visual representation of the inner-workings of the system in real-time, without affecting the actual application.

4.2 Applications

The user will be able to interact with the system in numerous ways. We provide a library of complete system stacks that the user can compile and readily deploy onto the nodes. One such application is object detection and tracking. A person can walk through the testbed and the application will detect the intrusion and track the location of the person as he or she moves through the network. Other programs include temperature and acoustic sensing and data collection, as well as querying of data coupled with in-network aggregation.

We also provide a variety of network layer protocols, such as tree building and routing, gradient routing, point-to-point routing, and runtime binary image distribution. The user can plug-and-play these components with a high-level applications without having to worry about the underlying communication primitives. For example, the user can write the specification for a short program where nodes report their acoustic readings if they are over a certain threshold, and simply use the tree routing component for underlying communication.

The user can also specify pertinent information to be downloaded at the base station in order to analyze the performance of the application. For the above applications, this could mean examining the readings of the sensors to determine which nodes has detected the presence of an object, or viewing a continuous stream of acoustic readings to make sure those above the threshold are reported.

4.3 Optimizations

The unique characteristics of sensor networks make them different from wired networks, and even wireless networks. Consequently, we employ numerous optimizations as discussed in the previous architecture section. The demo allows the user to examine the effect of each of these optimizations on the specific application at hand by tracing through the execution with the online debugger.

5. REFERENCES

[1] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *In ACM SIGPLAN Conference on Programming Language Design and Implementation, 2003.*, 2003.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[3] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In First Symposium on Network Systems Design and Implementation (NSDI)*, Mar 2004.

[4] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking with distributed recursive query processing. In *ACM SIGMOD International Conference on Management of Data*, June 2006.

[5] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 75–90, New York, NY, USA, 2005. ACM Press.

[6] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: Extensible routing with declarative queries. In *ACM SIGCOMM Conference on Data Communication*, August 2005.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *Transactions on Database Systems (TODS)*, March 2005.

[8] A. Tavakoli, J. Taneja, P. Dutta, D. Culler, S. Shenker, and I. Stoica. Evaluation and Enhancement of a Unifying Link Abstraction for Sensornets. In *UC Berkeley Technical Report*, 2006.