Mining a World of Smart Sensors

[Extended Abstract]

Suman Nath[†,*] Amol Deshpande[‡,*] Phillip B. Gibbons[*] Srinivasan Seshan[†,*]

[*]Intel Research Pittsburgh [†]Carnegie Mellon University [‡]U.C. Berkeley

{sknath,srini+}@cs.cmu.edu, amol@cs.berkeley.edu, phillip.b.gibbons@intel.com

1. INTRODUCTION

Imagine driving towards a destination in a busy metropolitan area. While stopped at a traffic light, you query your PDA specifying your destination and criteria for desirable parking spaces (e.g., within two blocks of your destination, at least a four hour meter). You get back directions to an available parking space satisfying your criteria. Hours later, you realize that your meter is about to run out. You query your PDA to discover that, historically, meter enforcers are not likely to pass by your car in the next hour. A half hour later, you return to your car and discover that although it has not been ticketed, it has been dented! Querying your PDA, you get back images showing how your car was dented and by whom.

This scenario demonstrates the potential utility of sensor-based services such as a Parking Space Finder, Silent (Accident) Witness and Meter Enforcement Tracker. While several research projects [2, 3] have begun to explore the use of networked collections of sensors, these systems have targeted the use of closely co-located resource-constrained sensor "motes" [4]. In this paper, we describe a sensor network system architecture, called *IRIS* (*Internet-scale Resource-Intensive Sensor services*), based on much more intelligent participants. We envision an environment where different nodes on the Internet (standard PCs, laptops and PDAs) have attached sensors such as webcams (video cameras attached to Web-enabled devices). Any sensor-based service can retrieve information from this collection of sensors and provide service to users.

While webcams are inexpensive and easy to deploy across a wide area, realizing useful services requires addressing a number of challenges, including filtering large data feeds to retrieve useful information, answering distributed queries with reasonable response times, combining multiple sources of information, etc. Our goal in IRIS is to create a common, scalable software infrastructure that allows services to address these challenges in a manageable fashion. This would enable rapid development and deployment of distributed services over a worldwide network of sensor feeds.

MOBICOM'02, September 23–28, 2002, Atlanta, Georgia, USA.
ACM 1-58113-613-7/02/0009.

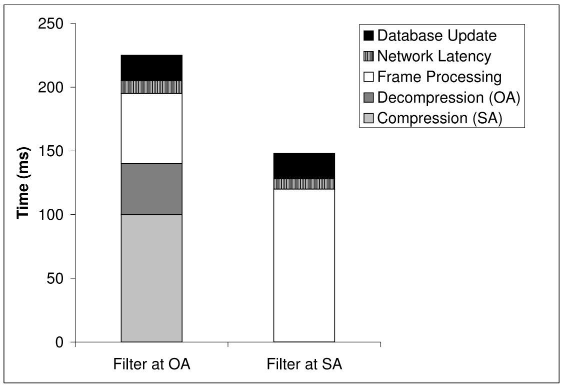

Figure 1: Sensor feed processing time

In this paper, we describe the design of IRIS and present initial results using a Parking Space Finder service. More information is available at `http://www.intel-iris.net`.

2. THE IRIS ARCHITECTURE

IRIS is composed of a potentially global collection of Sensing Agents (SAs) and Organizing Agents (OAs). Any Internet connected, PC-class device can play the role of an OA. However, less capable PDA-class devices can act as SAs.

Sensor-based services are deployed by orchestrating a group of OAs dedicated to the service. These OAs are responsible for collecting and organizing the sensor data in a fashion that allows for a particular class of queries to be answered (e.g., queries about parking spaces). The OAs index, archive, aggregate, mine and cache data from the SAs to build a system-wide distributed database for that service.

In contrast, SAs are shared by all services. An SA collects raw sensor data from a number of (possibly different types of) sensors. The focus of our design is on sensors, such as webcams, that produce large volumes of data and require sophisticated processing.

Key features of IRIS include:

- Providing simple APIs for orchestrating the SAs and OAs to collect, collaboratively process and archive sensor data while minimizing network data transfers.
- Handling issues of service discovery, query routing, semantic caching of responses and load balancing in a scalable manner for all services.
- Handling the demanding requirements of processing and querying live and historical high-volume sensor feeds such as webcams.

```
/usRegion[@id='NE']/state[@id='PA']
/county[@id='Allegheny']/city[@id='Pittsburgh']
/neighborhood[@id='Oakland' OR @id='Shadyside']
/block[@id='1']/parkingSpace[available='yes']
```

Figure 2: Query asking for all available parking spaces in Oakland block 1 or Shadyside block 1

2.1 Sensing Agents

OAs upload scripting code to any SA collecting sensor data of interest to the service. This scripting code programs the SA to take its raw sensor feed, perform the specified processing steps, and send the distilled information to the OA. An alternative design could be to send a generic compressed sensor feed from the SA to the OAs and to perform the filtering at the more powerful OAs. Our experiments have shown that the service-specific filtering of the sensor feed at the SA reduces the data transmitted from SAs significantly. For example, the Parking Space Finder only needs a few bytes to encode parking space availability instead of approximately 6KB per image frame for a webcam feed. Filtering at the SA also reduces the overall latency, as shown in Figure 1. The figure breaks down the time consumed for processing the sensor feed in each of the two designs, where the SAs are 550 MHz Pentium IIIs and the OAs are 1.6GHz Pentium IVs. Figure 1 shows that filtering at the OA is slower due to the overheads for compression and decompression.

A key concern is that because multiple services can share an SA, the more limited computing power of the SA may be overloaded. To alleviate this concern, our scripting code API allows downloaded code from different OAs to cooperate and avoid redundant processing. For example, both the Parking Space Finder service and the Silent Accident Witness service perform steps of filtering for motion detection and for identifying vehicles. We anticipate significant work sharing, because the downloaded codes are all specialized to the processing of the SA's particular sensor feeds.

2.2 Organizing Agents

Central to sensor networks is distributed query processing over streaming data [1, 5]. In IRIS, data such as the latest parking space availability information is stored in XML databases associated with each OA. Data for a particular service is organized hierarchically, with each OA owning a part of the hierarchy. (For example, a Parking Space Finder service may use a geographic hierarchy, e.g., usRegion–state–county–city–neighborhood–block). An OA may also cache data owned by other OAs.

A user's query, represented in the XPATH language, selects data from a set of nodes in the hierarchy. Figure 2 shows a typical XPATH query. The query contains a (maximal) hierarchical prefix, which specifies a single path from the root of the hierarchy to the lowest common ancestor (LCA) of the nodes potentially selected by the query. IRIS constructs a DNS-style name out of the prefix (e.g., for the query shown in Figure 2, IRIS constructs the name `pittsburgh.allegheny.pa.ne.parking.intel-iris.net`), performs a DNS lookup to get the IP address of this OA, and routes the query there.

Upon receiving a query, the LCA OA queries its local XML database, and evaluates the result. If necessary, it gathers missing data by sending subqueries to its children OAs (defined by the hierarchy), who may recursively query their children, and so on. Finally the answers from the children are combined and the result is sent back to the user.

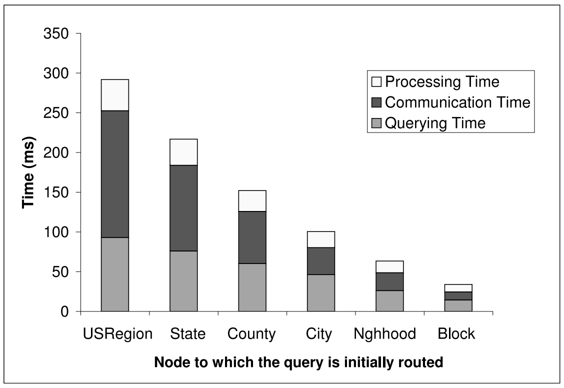

Figure 3: Query processing time

Figure 3 shows that, if the query is initially routed to a higher-level node in the hierarchy, most of the time is spent in network communication because of the high latencies between the upper-level nodes. As we come down to the lower-level nodes, the time taken to query the databases dominates the end-to-end query time. This also demonstrates the importance of intelligently routing a query directly as far down the hierarchy as possible.

In IRIS, queries may use the data cached in an OA, even if the new query is not an exact match for the original query. For example, the query in Figure 2 may use data for Oakland cached at the Pittsburgh OA, even though this data is only a partial match for the full query. The current prototype of IRIS supports a limited form of such *semantic caching.*

3. CONCLUSION

We believe that IRIS can enable a wealth of new sensor-based services. The next steps we are actively taking include investigating additional services, making the architecture more robust by handling both failures and graceful departures of nodes, improving performance through smarter semantic caching, and providing mechanisms for new SAs to be incorporated into active services.

4. ACKNOWLEDGEMENT

We thank M. Satyanarayanan for many valuable suggestions and C. Helfrich for helping with the experimental setup.

5. REFERENCES

[1] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Mobile Data Management*, 2001.

[2] D. Culler, E. Brewer, and D. Wagner. Berkeley Wireless Embedded Systems (WEBS). http://webs.cs.berkeley.edu/.

[3] D. Estrin, R. Govindan, and J. Heidemann. SCADDS: Scalable Coordination Architectures for Deeply Distributed Systems. http://www.isi.edu/scadds.

[4] J. Kahn, R. H. Katz, and K. Pister. Next century challenges: Mobile networking for 'smart dust'. In *MOBICOM*, 1999.

[5] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE*, 2002.