Dissecting the Video Sensing Landscape

Wu-chi Feng, Nirupama Bulusu, Wu-chang Feng
Department of Computer Science
Portland State University
Portland, OR 97207
{wuchi, nbulusu, wuchang}@cs.pdx.edu

ABSTRACT

Complex heterogeneous sensor networks that include both traditional scalar and richer video data are becoming possible. With such technologies comes the task of making them useful to the applications that rely on them. Unfortunately, such applications are extremely diverse in the interaction requirements of the system components. This paper argues that the system software for such sensor systems needs to balance optimization for performance and providing a flexible interface that can be tailored to the requirements of the user application. We propose *Cascades,* a cascading filter mechanism that can allow users to quickly compose and re-task diverse sensor systems (which include both scalar and video-based sensors) with highly optimized application-specific algorithms.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications

General Terms

Algorithms, Management, Performance, Design.

Keywords

Video sensors, sensor programming, sensor management.

1. INTRODUCTION

As sensor networking technologies continue to develop, a number of trends are starting to emerge. First, the diversity of sensor hardware is ever increasing, both in size and processing power. Smaller and more power efficient sensors are being developed to allow for more ubiquitous deployment. At the same time, "larger" sensors are being developed to support higher-level computation within the sensor network. Second, the diversity in sensing modalities is also increasing. The ability to capture a plethora of scalar data (such as temperature, sensor, and humidity) as well as data such as audio, images, infrared imaging, and video are becoming possible. Finally, the interaction and complexity between components within the sensor system will continue to increase. The interactions will most likely need to be tailored to a specific scientific application.

We believe that there are several common mechanisms that are still needed to support next generation sensor system software that include multimedia data types such as audio, imaging, or video. These include:

- Mechanisms to support diversity in sensor hardware, including those that capture non-traditional data such as images or video
- Mechanisms to easily program, deploy, and re-task diverse sensor networking technologies
- Mechanisms to support in-network, application-specific aggregation and adaptation of data

Unfortunately, many times the systems software and middleware deployed on such computing infrastructure is caught in the middle, trying to optimize the performance of the system while providing the flexibility, programmability, and abstractions needed by the applications.

In this paper, we argue for systems software that abstracts the hardware just enough to allow users to construct sensing systems with higher-layer abstractions that are composable and tailorable to a specific application's needs. We describe an architecture, which we term Cascades, that provides a number of properties to the application. First, it provides a high-level way in which application can specify the operation of the sensor system. This includes how the data should be managed and prioritized while it is being collected. Second, it allows the user to specify application-specific algorithms (optimized in the program language of their choice) to operate on the data within the network. For example, a particular sensor application may have a specific image processing algorithm or compression mechanism in mind. Finally, it provides a way in which heterogeneous sensors can be brought together while providing reasonable performance to the application.

NOSSDAV'05, June 13-14, 2005, Stevenson, Washington, USA.

In the following section, we briefly highlight the diversity and the systems software requirements to support such applications. In Section 3, we describe two sample video-based sensor networking applications. In Section 4, we propose our Cascades architecture. Section 5 provides an overview of the related work. Finally, we provide some directions for future research and conclude the paper.

2. THE VIDEO SENSING LANDSCAPE

In this section, we briefly describe some of the issues that need to be addressed for heterogeneous video sensor networking applications and the ramifications on the implementation of the systems software necessary to support the system. In particular, we provide the two extremes in which we believe the software will ultimately lie between.

Heterogeneity: The systems software will need to support a diversity of sensor hardware. To maximize efficiency, the systems software can export the bare minimum abstraction of the underlying hardware (e.g. TinyOS) but making it hard to manage, program, and connect with other hardware in an application-specific way. At the other extreme, one could provide a virtual machine interface to all hardware [9]. This approach makes the programming and management of the system more efficient. Given its higher overhead, however, it may not even be possible to push the abstraction to the smaller devices. Furthermore, providing virtual machine abstractions may be very inefficient for large data streams such as video because performance features such as memory mapping I/O devices may not be allowed.

Composability: Undoubtedly, the actual operation of the system will need to be tailored to a specific application's requirements. We expect that the sensor system will provide (i) a number of pre-defined components that can be used, (ii) mechanisms to support the addition of new components, and (iii) the ability to combine the components in a meaningful way. At the one extreme, composability can be accomplished through pre-defined code segments that are compiled together into a single monolithic executable, allowing the system to run as efficiently as possible, while making changes to a running system more difficult. At the other end of the spectrum, one could imagine using a shell-level scripting program to compose such as system together from a number of smaller executables. While making it easier to distribute smaller subcomponents, the system may suffer from a large amount of overhead in switching between address spaces and marshalling of data between executables.

Adaptability: The system will need to adapt to a number of conditions including the available computation, networking availability, and power. For example, an embedded device may be able to capture video but may be constrained in its ability to compress or transmit it. To provide the most efficient operation, one could tune the sensor software to capture, compress, and transmit as much information as the smallest bottleneck in the system can handle. Furthermore, one could hard code exactly how the system should respond to a number of external events such as network congestion and variable power generation (e.g. solar panel). Clearly such a system would be hard to retask or specialize to a new application. At the other end of the spectrum, the individual components could "self-adapt" or infer the amount of data that ought to generated, stored, or thrown away through indirect measures (e.g. the network buffer is getting full). While such a system may not be optimal in its operation, it is easier to maintain and tune in place.

System performance: The optimization of data flow through a sensor can have a tremendous impact on its power usage and its performance. As an example, consider the Panoptes video sensor [2]. Using a plethora of optimizations including memory mapping the camera device into the address space, using compression across the USB 1.0 interconnect, and using the Intel Performance Primitives, the sensor is able to achieve the capture and compression of approximately 24 320x240 frames per second on a 3 Watt, 200-MHz embedded device. The Intel Performance Primitives nearly tripled the frame rate achievable for DCT-based video compression. These primitives, however, are designed for only one processor architecture. As a result, one can highly optimize the code for a particular hardware platform and camera combination, but it may not be suitable for any other hardware and camera combination. Using generic interfaces without much optimization yields only a handful of frames per second with nearly the same code.

Clearly, the creation of next generation sensor system will need to manage the conflicting goals of performance and management. We will propose an architecture for such a system in Section 4.

3. DIVERSITY IN SENSOR NETWORK APPLICATIONS

To motivate our architecture, we briefly highlight two sensor networking applications that require heterogeneous sets of sensors that include video.

Advanced Health Care Delivery: Biomedical Engineers are interested in understanding the causes of the onset of dementia in elderly people. The goal of their work is to record the movements and actions of the subjects in-situ, combine it with other data such as weight and vital signs, and use this to detect long-term changes in behavior and the possibility of the onset of dementia [3]. With early detection, the quality of life of a person can be extended while simultaneously reducing the cost of care.

To capture macroscopic actions, the researchers use arrays of scalar-based sensors to capture the actions of the subjects. Such data is fraught with errors from trying to distinguish the subject from other objects such as their pets. In this context, a single video-based sensor can augment the scalar sensors to help determine the presence of the targeted resident. The video sensor will allow for actions to be more easily and accurately captured for the scientists. This, however, requires a

mechanism by which they can easily combine both sensor and video data in an application specific way. For example, to measure gait speed, the scalar sensors may be used to accurately measure the resident's speed, while the video sensor may be used to distinguish between humans and pets. In the same home, a video sensor may be used to record gestures or actions (e.g. cooking in the kitchen).

Environmental Monitoring: Environmental scientists and oceanographers are interested in the evolution of near shore phenomena along the coastal margin. The oceanographers have developed techniques to use imaging data to understand the evolution of sandbars underneath the water's surface off of the coast. While they could use a massive array of in-water scalar sensors, one video sensor and scalar environmental readings can provide the same information for their research with significant infrastructure cost savings, ease of deployment, and reduced maintenance costs [10].

Currently, their video sensor deployment requires power and networking close by. They would ultimately like to install a video-based sensor every quarter mile along the entire Oregon coast, requiring some 1200 sensors networked together, surviving on power generated from the coastal winds or solar power. This will require both advances in basic sensor networking technologies as well as power-adaptive video capture and transmission mechanisms. In addition, they would like a mechanism to coordinate several sensors together to capture events such as rip currents.

4. CASCADES: A HIGH-LEVEL, COMPOSABLE FILTERING AND ADAPTATION INFRATRUCURE

We propose an architecture to support the diversity of sensor networking applications, while providing reasonable systems-level performance. We considered a number of options using the issues we outlined in Section 2 in mind. Sensor code compiled into one executable was ruled out because updating the functionality of a video sensor would require a significant amount of wireless bandwidth to be used to distribute all the code, whether it changed or not. At the other extreme, one could use shell-level scripting, connecting individually compiled pieces of code to be brought together. This solution was also ruled out as not having high enough performance. The other two alternatives were a high-level scripting language like TCL or a lower-level scripting language like Python. The key advantage of such languages is that they are interpreted scripting languages, allowing users to specify rather complex systems with minimal code. Furthermore, they allow programs written in high-level languages such as C or C++ to be called as part of the script. This allows a majority of computationally intensive code (such as video processing algorithms) to be written in a highly optimized way.

TCL-based multimedia systems have been built and demonstrated. The Continuous Media Toolkit (CMT) from Berkeley is one such example [6]. Using their CMT API, one can create a streaming video session with relatively few lines of code (< 10 for a streaming video-only application). The key here is that much of the functionality in the CMT toolkit is nicely encapsulated into a number of objects that can then be put together through the TCL script. We believe that for the low-power video sensor networking applications the functionality to manage and filter data needs to be pushed as far to the edge of the network as possible. As this functionality needs to be tailored to the application, a slightly lower level mechanism to dynamically connect components together was needed.

Our architecture uses Python-based interfaces to connect filters together. The basic concept of building a system out of TCL and Python are similar. We chose Python for several reasons. First, it provides more complex data structures than TCL. Second, it is more efficient than TCL, which is extremely important for the power-constrained sensor systems. Third, Python provides the ability to add or change the behavior of parts of the system while it is running. As a result, for the parts of the system that are connected via Python, retasking the system involves distributed the new code segment and updating the script so that it points to the new code (e.g. a new video compression algorithm)[1]. Fourth, Python interfaces also provide the opportunity to provide type checking of the data so that the components that are plugged together can be verified for compatibility. Finally, it is easier to construct more complicated programs in Python, giving the user more control over the system rather than hiding many details. The last point is both a positive and negative. In the hands of more experienced programmers, Python is easier to adapt to application specifics. We have implemented the video sensor code to have its various components (capture, filter, compression, and buffering) connected via Python. The Python script for this is approximately 150 lines of code. Obviously the code is somewhat more complex than the type of system specified in the CMT API, however, it provides much finer grain control of the code within the sensor.

4.1 Constructing Cascades

The primary mechanism to support the management and integration of heterogeneous data are cascading filters. This basically extends the functionality that we have implemented in the video sensors. Filters are user-supplied or toolkit-derived functions that allow the sensor system to tailor its data for the user application. The idea of each filter is that it allows the processing of the data within the filter to be accomplished with a highly optimized piece of code (rather than an interpreted language). There are several basic filters that we envision:

- *Efilters* are the primary mechanism by which the handling of faulty sensors can be specified. Faulty readings can occur from biofouling of the sensors in outdoor scenarios. These

[1] This assumes that the Python script periodically checks whether or not the script has been updated.

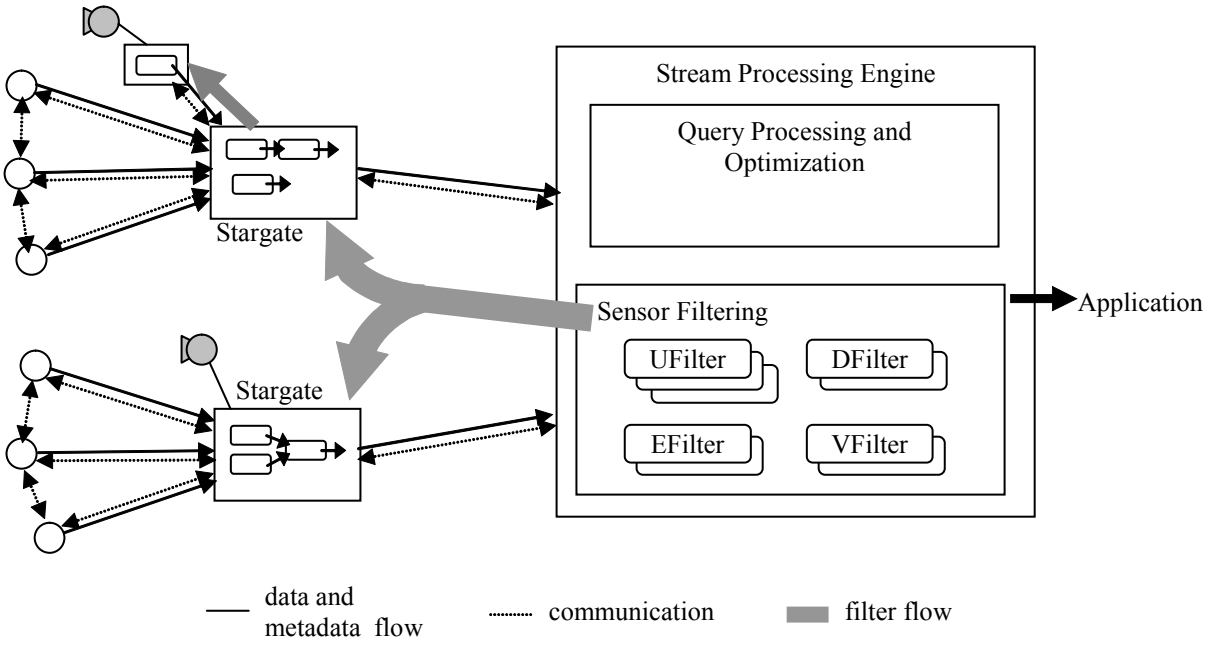

FIGURE 1: This figure shows an example of the overall architecture for our proposed sensor networking middleware. The nodes labeled *Stargate* are slightly more powerful, in-network, sensor nodes that can both capture video and be used to manage a number of scalar sensor nodes.

filters can consist of standard statistical filtering techniques in a default-model as well as for the application user to specify the exact way in which the faulty data may be handled.

- *Dfilters* are used to manage scalar data within the sensor network. They take one or more streams of scalar data and produce an output of one or more data streams as well as meta-information about the sensor data. As an example, one filter might calculate the average value measured per hour, either for a single sensor or a group of sensors. The filter might also add meta-information such as timing information or relational information between sensors. The sensor output can then be used by other filters. As we will describe later,
- *Vfilters* are used to manage video data being collected by video sensors. Vfilters might consist of application specific video processing algorithms or off-the-shelf components. Application-specific algorithms may include image processing techniques for the environmental monitoring example previously described. An off-the-shelf component might include a compression algorithm or video adaptation algorithm within the network.
- *Ufilters* are user specified filters that allow the user to specify the integration of data from the other types of filters. These can include annotation of video streams using scalar sensor data.

The overall Cascades architecture is shown in Figure 1. We expect that the filters can then be cascaded together using Python-based interfaces between filters. The actual hardware devices are abstracted just enough to provide data into one of the filters. For example, scalar data sensors will have the operating system of their choice running on the system and will talk upstream to a node (such as a Crossbow Stargate device) that will provide the Python-based abstraction. For video sensors, the camera device is abstracted enough to provide access to the raw frames.

4.2 Python Details

4.2.1 Marshalling Data in Python

While Python provides a great deal of flexibility in its structure and the ability to dynamically alter the algorithms running on the sensor, it does incur some overhead. This overhead is primarily in the marshalling of the data across the interfaces. Because we are trying to build sensors that can be easily composed together, the interfaces need to be necessarily generic in nature, providing a "least common denominator" handling of the data.

We have implemented the key components within the video sensor software to be connected via Python. The components we currently have specified are video capture, video filtering, video compression, and video adaptation. Between each component, we have created *data_frames* that each hold information regarding one frame of data. Because each component understands what a *data_frame* is, composing new

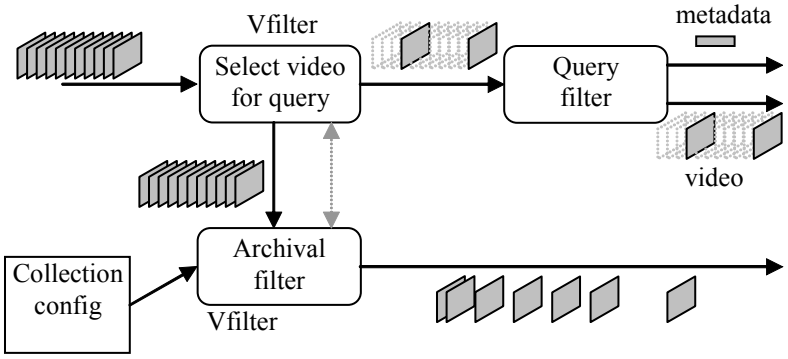

Figure 2: This figure shows an example filter mechanism we expect to be able to support. The top video filter selects video necessary for the query (either as a result or input to another filter). The query filter generates metadata and passes on the video images. The lower archival filter is responsible for archival of the video data and is responsible for thinning the stream if insufficient resources exist. Other equivalent filters are possible. For example, the user may combine the query filter and the selection filter at the top.

compression or image processing algorithms is fairly easy. Our experiments have shown that the overhead of Python in this context ranges from 3-5%. We believe that this is reasonable, given the additional flexibility that is gained.

4.2.2 Constructing Complex Systems

We have recently demonstrated a video sensor based upon the low-power (< 2 Watt) Crossbow Stargate device. The Stargate device is meant to act as a gateway for low level sensors such as the motes. It has a 400 MHz Intel XScale processor, 64 Mbytes of memory, and runs at approximately 2 Watts. Python and the video sensor compression code easily fit within the memory available on the device.

Unfortunately, Python is too large to run on the scalar sensors which typically have only kilobytes of available memory. In order to construct larger systems together, we believe that such systems will ultimately have a multi-tiered architecture that have both low-level sensors and higher-powered sensors (e.g. video sensors) that are capable of aggregating data. We believe that the scalar sensors can be abstracted to the point that they can be plugged, as input, into the Cascades systems. Furthermore, the scalar sensors can be brought together with traditional data gathering techniques (e.g. TinyDB or Directed Diffusion) if so desired. Finally, as the Crossbow Stargate devices are meant to bring such devices together, we expect that providing basic abstraction over the smallest devices will not adversely affect system performance.

4.3 AN EXAMPLE SYSTEM

As a small example of our envisioned system in use, suppose we have a sensor system that processes video data, creates metadata required to answer a user query, and attempts to archive as much of the video for storage as it can. In this scenario, a filter may create metadata to answer a stream query. This may also involve the transmission of part or all of the video for the period to the upstream node for processing of the upstream query. We have shown a sample filter flow that we would expect to use in Figure 2. To support such data management, however, the user or application needs to specify how and what data should be dropped in the event that insufficient network or power resources exist to fully capture the entire stream. We expect that this will be highly application specific. Of course, many other types of video management policies may need to be implemented for a particular application and the middleware ought to be flexible enough to support such policies. We envision that the specification of *how* the video data should be prioritized and dropped will be specified within the Python script. The specification of *which* data is required by various stream queries will be accomplished through the filters that we have previously described. Note that the video filters shown can be augmented with scalar sensor data to turn on and off video processing.

One thing we have not yet addressed is the coordination among various filters. Without distributed coordination, duplication of data may occur if the system is not carefully constructed.

5. RELATED WORK

Several sensor platforms have been developed with varying capabilities for use in sensor networking applications, including the Berkeley mote family of WeC, Rene, Dot, Mica, MicaZ, and XSM sensors [7], image-based sensors such as Cyclops (based on the Mica2 sensor) [8] and the Panoptes video sensor (based on the Crossbow Stargate device [2]. These developments just highlight the fact that the systems software will have to evolve to support a diversity of sensors.

Techniques such as Directed Diffusion [5] or TinyDB [4] are used to provide aggregation for traditional scalar-based sensor networks. In contrast to Directed Diffusion, Cascades enables a whole-network reconfigurable deployment of filters, instead of pre-deployment programming of individual sensors. Thus, Cascades can provide better programmability and aggregation. Cascades shares the programmability objectives of TinyDB, but can process data streams and task sensor nodes in more flexible ways. These distinctions are significant because they enable us to support a more diverse set of sensor devices while also providing programmability and flexibility. Cascades can also be used with Diffusion or TinyDB. For example, one can control a herd of motes via TinyDB, and use TinyDB's data stream output as an input to a Cascade filter

There are many stream query processing systems for sensor networks including Cougar, Telegraph, Stream, and Aurora/Borealis [1]. One key difference between the existing stream processing work and Cascades is the inclusion of video processing and filtering within the sensor substrate as well as application-specific data handling and adaptation in the event of insufficient resources exist. We believe that much of the functionality of such stream processing engines can be incorporated into the filtering mechanisms we propose.

Finally, we note that finding efficient "plug and play" architectures for multimedia has been the focus of some previous research. The *Continuous Media Toolkit* (CMT) from Berkeley focuses on the rapid development and deployment of distributed streaming applications [6]. The CMT toolkit is a Tcl/Tk-based system that allows the users to construct streaming applications through scripts that combine lower-level components together. The Cascades approach is similar in vein to the CMT toolkit. The sensor networks we envision, however, will force the scripting languages to be more complex in it management of data between components. We believe that Python is more usable in this context because it provides real data structures to the scripts and runs faster than Tcl. The data structures are necessary to support the movement of data between the filters.

6. ACKNOWLEDGEMENTS

The authors would like to thank Edward Epp of Intel for making the Stargate platform usable for Video for Linux and the Logitech cameras, which our video sensors use. This material is based upon work supported by the National Science Foundation under Grant No. EIA-0130344. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. CONCLUSION AND FUTURE WORK

In this paper, we have outlined a potential architecture to support diversity and programmability in heterogeneous sensor networks that include both scalar and video data. As sensor networks continue to evolve, the systems software that supports such applications needs to evolve while continuing to provide highly optimized and efficient operation. Python-based cascading filters can effectively manage the trade-off of highly optimized systems and the need for to tailor the system to the user application.

There are still a number of open issues that we are working to address with Cascades. Currently, there is no explicit mechanism to adapt computation to conserve power. Unfortunately, this requires coordination among filters. Another avenue of future work is managing power on a larger time-frame. For example, in the oceanographic example, it may be more beneficial to store data during the day and only transmit at night (while no video capture is happening).

8. REFERENCES

[1] D. Abadi, et. al, "The Design of the Borealis Stream Processing Engine", in *Proc. of the Second Biennial Conference on Innovative Database Systems (CIDR'05*), January 2005.

[2] W. Feng, B. Code, E. Kaiser, M. Shea, W. Feng, L. Bavoil, "Panoptes: A Scalable Architecture for Video Sensor Networking Applications", in *Proc. of ACM Multimedia*, Nov. 2003, pp. 562-71.

[3] T. Hayes, M. Pavel, P. Schallau, AM Adami, "Unobtrusive Monitoring of Health Status in an Aging Population", in *Proc. of the 5th Inter. Conf. on Ubiquitous Computing*, Oct. 12-15, 2003.

[4] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", in *Proc. of the Sixth Annual MobiCOM*, August 2000.

[5] S. Madden, M. Franklin, J. Hellerstein, W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", in *Proc. of OSDI*, Dec. 2002.

[6] K. Mayer-Patel, L. Rowe, "Design and Performance of the Berkeley Continuous Media Toolkit", in *Proc. of Multimedia Computing and Networking 1997*, pp 194-206, 1997

[7] J. Polastre, Design and Implementation of Wireless Sensor Networks for Habitat Monitoring, M.S. Thesis, EECS, Univ. of Calif., Berkeley, 2003.

[8] M. Rahimi, D. Estrin, R. Baer, H. Uyeno, J. Warrior, "Cyclops: image sensing and interpretation in wireless networks", in *Proc. of ACM SenSYS 2004*, November 2004.

[9] Sense Project: http://senses.cs.udavis.edu

[10] H.F. Stockdon, R.A. Holman, "Estimation of Wave Phase Speed and Nearshore Bathymetry from Video Imagery", *Journal of Geophysical Research*, Vol. 105, No. C9, pp 22,015-22,033, Sept. 2000.