ViRe: Virtual Reconfiguration Framework for Embedded Processing in Distributed Image Sensors

Rahul Balani, Akhilesh Singhania, Chih-Chieh Han and Mani Srivastava
University of California at Los Angeles
Los Angeles, CA 90095, USA

Abstract—**Image processing applications introduce new challenges to the design of sensor network systems via non-trivial in-network computation. As embedded processing becomes more complex, in-situ reconfiguration is seen as the key enabling technology to maintain and manage such systems. In dynamic event-driven heterogeneous sensor networks, reconfiguration also encompasses autonomous re-partitioning of applications across multiple tiers to provide a low-power responsive system by efficiently coping with variations in run-time resource usage and availability. Hence, we aim to provide an efficient low-power macro-programming environment that supports multi-dimensional software reconfiguration of heterogeneous imaging networks.**

Working towards this initiative, we present the ViRe framework for mote-class devices based on data-centric application composition and execution. Applications, modeled as dataflow graphs, are composed from a library of pre-defined and reusable image processing elements. Concise scripts capture the wiring information and are used to install applications in the network, while execution on the nodes is performed via processor native code to minimize overhead. A lean run-time engine tightly monitors application execution to provide an efficient, robust and scalable support for complex reconfigurable embedded image processing. Thus, the system is able to lower application re-partitioning overhead and minimize loss of work during software reconfiguration.

Index Terms — **Sensor Networks, Differential partitioning, Reconfiguration, Dataflow**

I. Motivation

Image sensors demonstrate vital importance in understanding and characterization of diverse environments. Applications involving these sensors pose new challenges to the design of sensor network systems, particularly in the context of recently developed low-power, but resource-constrained, image sensing platforms such as Cyclops [1] and AER Imager [2]. In-network sensor information processing takes on a particularly important role with image sensors, because to save communication bandwidth and energy, image sensor nodes must process the images *in-network* to extract relevant events or features before transmission. Prior work, comprising of mere filter chains, linear dataflow graphs or SQL aggregate functions, is not enough to affect adequate data reduction. Rather, significantly richer and complex recursive algorithms are needed to compress image frames, or to detect features or events of interest. Another reason for the recursive nature of the computation is that the high cost of image acquisition itself usually motivates application structures where a feedback control loop controls when to acquire the next image frame based on history. Thus, typical image processing algorithms require system software support for *efficient* and *reliable* functioning when embedded in resource-constrained sensor nodes.

Clearly, the ability to *reconfigure* application-specific embedded sensor processing on the nodes at run-time, for purposes of re-tasking and environment-specific tuning, without sacrificing the efficiency of processing, is important for effective operation and maintenance of complex sensor networks. This recognition has led to emergence of systems such as Contiki [3], SOS [4], TENET [5], VanGo [6], and Maté [7] where run-time retasking and reconfiguration of application-specific processing at the node are directly supported. However, these prior systems are either too low-level (e.g. Contiki and SOS) and thus do not provide support for higher level programming abstractions common to image sensing, or significantly restrict the complexity or efficiency of application-specific processing and flexibility of reconfiguration (e.g. TENET, VanGo, and Maté) and thus unable to support the requirements of reconfigurable embedded image processing.

However, in a dynamic event-driven multi-tier heterogeneous sensor network, software reconfiguration is not restricted to simple parameter and logic updates, but involves dynamic re-partitioning of the data processing pipeline across different tiers. It is required in dense high data-rate networks that exhibit a clear trade-off between local computation and transmission of data in terms of both latency and energy. For instance, processing data locally at a certain sensor node may reduce energy consumption by reducing data transmission across the network, but may increase overall time (latency) required to obtain the result at the base station due to lengthy computation. This trade-off is unique to each node as the network latency experienced at that node is not only affected by its proximity to the intermediate or final data sink, but also by status of its local transmission queue and network traffic at higher tiers. Thus, to support often conflicting goals of low latency and low power, it is necessary to monitor the above factors at run-time and migrate computation appropriately, resulting in different configurations of the same end-to-end pipeline at different nodes. Henceforth, this dynamic migration of components of a data processing pipeline, to support application re-partitioning, is referred to as *task migration* in this text.

Prior work [8][5] has essentially focused on providing an efficient, robust and scalable macro-programming environment for heterogeneous sensor networks, but consistently ignored the complex dynamics of an event-driven network and software reconfiguration in the broader sense of run-time repartitioning of applications. Hence, we propose to build an efficient, robust and reconfigurable macro-programming framework for heterogeneous sensor networks that autonomously monitors dynamic network context and supports application repartitioning via task migration to achieve user-specified goals.

II. Introduction

As the first step in achieving our objective, we present the ViRe (Virtual Reconfiguration) framework for mote-class devices that constitute the lowest tier of a heterogeneous sensor network (figure 1). Motivated by results from our previous work [9], it explores a different point in the design space of retaskable and reconfigurable embedded sensor processing. It exposes a visual modular wiring diagram abstraction that is commonly used to express image processing algorithms. The modules encapsulate image processing functions, while the wiring is used to express dataflow which may be *nonlinear* and with *feedback loops* (figure 3). ViRe permits wiring, module code, and module parameters to be incrementally reconfigured.

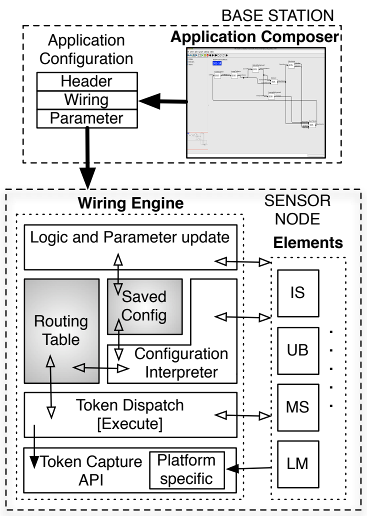

Fig. 1: ViRe Framework

Main idea behind this representation is to separate application instantiation on the nodes, via wiring information captured in concise *scripts*, from execution and data exchange performed in processor native code via dynamic linking at load-time. It has multiple advantages: (i) Execution overhead is kept low while update costs are reduced dramatically, (ii) Work done on prior data is conserved through unobtrusive application of incremental updates, referred to as *hot-swap*, and (iii) Migration overhead is reduced significantly as only a part of the wiring script is required to be transferred across the network during application re-partitioning.

Target sensor nodes interpret the *script*, generated by the *application composer*, to install or update the application through the resident run-time *wiring* engine. Basic communication and scheduling abstraction of the engine, a *token*, provides for efficient management of image data through cooperative memory sharing at run-time. ViRe optimizes the handling of image data since copying an entire sample (image frame), or worse a full block of multiple samples, between processing functions would be prohibitively expensive requiring multiple reads and writes of large, and typically off-chip, frame-buffer memory [1].

The *Token Dispatch* mechanism in the wiring engine ensures that execution follows correct data-flow semantics by coordinating the passage of tokens between elements. This coordination provides explicit control to the engine over application functioning and promotes concise element implementation, thus attributing to the reduced update cost. It is exploited to enable recovery from execution errors, protect against race conditions due to feedback, facilitate a *safe* hot-swap during graph update and minimize transfer of state during task migration.

Application execution begins when new input data is generated by the source element. An element is *fired* whenever its ready to accept new input on a port and a token is placed on that port. Depth-first traversal is followed where the constituting elements run to completion unless they yield, either waiting for inputs on multiple ports or for allowing the engine to schedule other tokens. Consequently, token queues are maintained on graph edges by the engine, instead of the elements, to promote concise module implementations. Thus, an application can be mapped to a pipeline architecture for analysis of its asynchronous behavior and various flow-control strategies at run-time.

The ViRe framework is currently implemented on top of SOS operating system [4] for the cyclops platform [1]. Cyclops is built as an imager sensor board for mote class devices like the Mica motes. Besides other components, it consists of a CMOS imager, an AVR AtMega 128L micro-controller, and 60 KB of external SRAM that is mainly used for buffering images.

III. Wiring Engine: Design

Supporting complex and reconfigurable data-flow representation on the sensor nodes introduces several challenges, including how to link the elements dynamically at run-time to support multiple fan-ins and fan-outs, and efficient exchange of data; how to avoid race conditions due to feedback in recursive image processing algorithms; and how to detect errors in the system to provide reliable functioning. In addition, handling high rate data requires efficient sample processing and memory utilization during execution. The ViRe run-time addresses all these challenges to provide a reliable system for reconfigurable embedded image processing with minimal memory and execution overhead.

A. Application Installation: Configuration Interpreter

Intelligent application installation is necessary to support fast execution with minimal memory overhead. The wiring engine uses a concise routing table, supported by a clear element

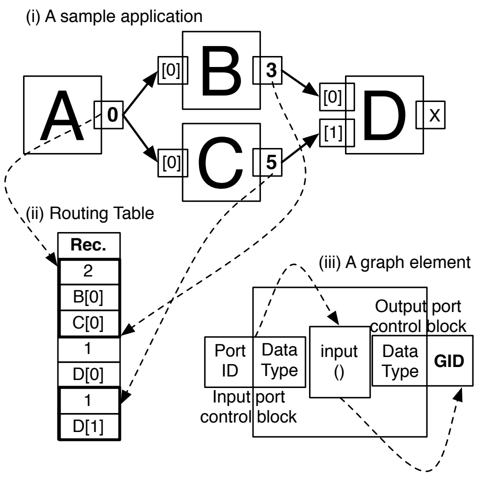

Fig. 2: A sample application graph, its corresponding routing table and element design showing its input and output control blocks.

design as shown in figure 2(iii), to represent the graph edges. The routing table is constructed from the configuration *script* and consists of a group of *output port records* representing the corresponding fan-out of each element port. A record is used whenever an element places a token on its port to facilitate data transfer. The ViRe engine accesses the record through an application-specific unique identifier (GID) that acts as a direct index into the routing table and is assigned to each output port at installation/load time. This is an inexpensive constant time access that helps achieve low overhead during data exchange as discussed in section II. Finally, to protect against run-time failures, the data types associated with input and output ports are used at application initialization to validate their dynamic linking.

B. Application Execution: Token Dispatch and Token Capture API

Execution of image processing algorithms in ViRe framework consists of *application specific computation* and *communication* between graph elements. Communication between elements involves transfer of data wrapped in a token structure [10]. It simplifies management of data by providing support for tracking its read-write permissions, ownership and platform specific type. This information is used by the **token capture API** to allow elements to create tokens and cooperatively share memory by releasing and capturing them.

The **token dispatch** mechanism coordinates the exchange of data between graph elements to facilitate efficient communication. Access to an input port of a destination element is enabled via a synchronous function call that provides the input token to the element and returns its current status to the engine. This implicit interaction enables the engine to tightly monitor application execution, maintain token queues for the elements when they are busy processing other tokens, and detect and recover from run-time errors. Thus, an *output port record* is constructed through dynamic linking [4] of multiple callee functions (input ports) to a single caller (output port) and storing the pointers for fast run-time access.

Synchronous data exchange is chosen over asynchronous message passing for execution and memory efficiency, and enabling the engine to control application execution. During execution, it results in a complete traversal of the sub-graph rooted at the output port, referred to as one *output port iteration*. Hence, to avoid race conditions due to feedback in the same iteration, the element is marked busy, and all input tokens destined towards it are queued.

C. Application Reconfiguration

The wiring engine supports logic and parameter reconfiguration on the embedded target, as well as task migration in a tiered sensor network. Logic reconfiguration involves (i) major updates that include significant modifications to the graph edges and elements such that the application, or its implementation, changes as a whole, or (ii) minor updates that include small changes to the graph like addition or removal of a wire or an element. Installing a major update removes the current application completely and initializes a new application as described in section III-A. The rest of this section focuses on the support for applying minor updates through *hot-swap* and reducing state transfer during task migration.

A mechanism supporting hot-swap should minimize loss of work on previously processed tokens and respect data dependencies between elements. Partially processed data, that may be relevant after update, should not be discarded. Data loss is only possible when an *active* element is replaced during the update leading to loss of token(s) being processed by the element at that time.

Typically, task migration across network involves code and state migration. Code size is already minimized through concise wiring representation as discussed earlier. The state of the element, which needs to be migrated across the network, should be minimized to reduce migration overhead. We can observe that local state of the element is minimum when it is not *active* as it does not contain information on partially processed token(s).

Thus, the engine hot-swaps updates or migrates tasks only when none of the involved elements are *active*. But, it is possible that the involved elements, though not *active*, may contain application pertinent state. Currently, the state of the old unused elements is discarded, i.e. not migrated to the new elements that may have replaced them. Therefore, hot-swap and task migration are suggested to include only the elements that never contain application pertinent state. However, in future versions, we propose to extend this mechanism to migrate old state across such incremental graph replacements and element migrations.

IV. Evaluation

This section provides a brief evaluation of the ViRe framework in terms of its update costs and execution overhead on a resource constrained micro-controller like AVR Atmega128.

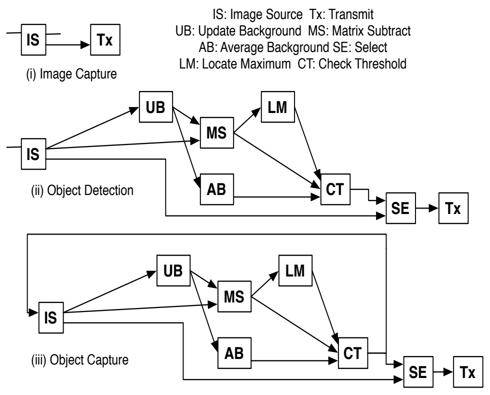
IS: Image Source Tx: Transmit
UB: Update Background MS: Matrix Subtract
AB: Average Background SE: Select
LM: Locate Maximum CT: Check Threshold
(i) Image Capture
(ii) Object Detection
(iii) Object Capture
IS
Tx
UB
MS
LM
AB
CT
SE

Fig. 3: Application Graphs

Applications	Exec. time *w/out ViRe*	ViRe Overhead	
		Mem. Opt.	Exec. Opt.
Image capture	82 μs	240 μs	80 μs
Object Detect	136 ms	5.6 ms	3.2 ms
Object Capture	136.2 ms	5.8 ms	3.3 ms

TABLE I: Total execution time
Memory optimized and Execution optimized versions of ViRe

Allocation Type	**Component**	**Memory** (bytes)
Static	Wiring Engine	36
Semi-Dynamic	Graph Elements (RAM)	41
	Routing Table (Mem. Opt. - Flash) (Exec. Opt. - RAM)	IC - 6 OD - 57 OC - 60

TABLE II: Memory Allocation
IC - Image Capture, OD - Object Detect, OC - Object Capture

Using the successive versions of surveillance application (figure 3) discussed in [10], we demonstrate that the system reduces logic reconfiguration cost of the application by at least an order of magnitude as compared to native implementation in SOS. This is attributed to concise wiring representation and aggressive reduction in the size of image processing modules by promoting simplicity in their design.

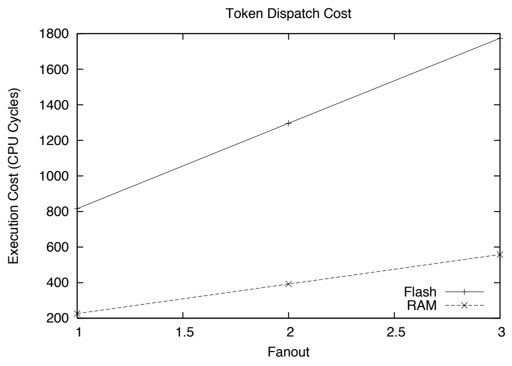

Fig. 4: Communication cost (Token dispatch) at an output port as a function of its fanout when the routing table is stored in RAM vs Flash memory

Moreover, this dramatic decrease in update cost incurs a low execution overhead (table I) ranging from mere 240 μs to 5.8 ms depending on the structure of the application graph. Table II shows the concise memory footprint of the ViRe framework along with installed applications. A thorough analysis of the execution framework [10] helped in identifying and removing communication bottlenecks, thus causing an average 50% reduction in the above overhead at a small expense of increased memory consumption (table I, column *Exec. Opt.*). Figure 4 establishes that the communication cost associated with the framework is closely tied to the graph topology i.e. the number of output ports and their fan-out. Our work in [10] further demonstrates that this cost is independent of the size of data exchanged between elements.

V. Future Plans

Finally, we believe that ViRe overhead will be negligible on processors like Intel/Marvell X-scale, popularly used in higher tiers of sensor network deployments, due to a richer instruction set, faster clock and larger memory. Next, we plan to extend this framework to a heterogeneous multi-tier network and implement distributed network monitoring to investigate the impact of network conditions on autonomous application partitioning that aims to balance low latency goals with minimal energy consumption. Preliminary results from a similar high data-rate network [11] show that the latency can be decreased by as much as 50% through intelligent application partitioning. Later, we will extend it to include the effect of energy availability in an energy-harvesting network.

References

[1] M. Rahimi, R. Baer, O. Iroezi, J. Garcia, J. Warrior, and M. Srivastava, "Cyclops: in situ image sensing and interpretation in wireless sensor networks," *EmNets*, pp. 192–204, 2005.
[2] T. Teixeira, E. Culurciello, J. Park, D. Lymberopoulos, A. Barton-Sweeney, and A. Savvides, "Address-event imagers for sensor networks: evaluation and modeling," *IPSN*, pp. 458–466, 2006.
[3] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," *EMNETS*, 2004.
[4] C. Han, R. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava, "Sos: A dynamic operating system for sensor networks," *Mobisys*, 2005.
[5] O. Gnawali, K. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler, "The tenet architecture for tiered sensor networks," *Sensys*, pp. 153–166, 2006.
[6] B. Greenstein, C. Mar, A. Pesterev, S. Farshchi, E. Kohler, J. Judy, and D. Estrin, "Capturing high-frequency phenomena using a bandwidth-limited sensor network," *Sensys*, pp. 279–292, 2006.
[7] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *SIGOPS*, vol. 36, no. 5, pp. 85–95, 2002.
[8] A. Awan, S. Jagannathan, and A. Grama, "Macroprogramming heterogeneous sensor networks using cosmos," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, 2007.
[9] R. Balani, C. Han, R. Rengaswamy, I. Tsigkogiannis, and M. Srivastava, "Multi-level software reconfiguration for sensor networks," *EmSoft*, pp. 112–121, 2006.
[10] R. Balani, A. Singhania, C. Han, R. Rengaswamy, and M. Srivastava, "Vire: Virtual reconfiguration framework for embedded processing in distributed image sensors," NESL, UCLA, Tech. Rep., June – October 2007.
[11] M. Allen, L. Girod, R. Newton, D. Blumstein, and D. Estrin, "Voxnet: An Interactive, Rapidly-Deployable Acoustic Monitoring Platform," *SPOTS*, 2008.