

# Symbiote - A Reconfigurable Logic Assisted Data Stream Management System (RLADSMS)

Pranav S. Vaidya, Jaehwan John Lee,  
Francis Bowen, Yingzi Du  
Electrical and Computer Engineering  
Indiana Univ. Purdue Univ. Indianapolis  
[psvaidya, johnlee, frbowen,  
yidu]@iupui.edu

Chandima H. Nadungodage, Yuni Xia  
Computer and Information Sciences  
Indiana Univ. Purdue Univ. Indianapolis  
chewanad@iupui.edu  
yuxia@cs.iupui.edu

## ABSTRACT

Numerous monitoring applications such as traffic control systems, border patrol monitoring, and person locator services generate a large number of multimedia data streams that need to be analyzed and processed using image processing and data stream management techniques in order to detect significant events of interest or abnormal conditions. Such multimedia monitoring systems usually have high-bandwidth characteristics, stricter real-time deadlines, and high accuracy requirements. In an attempt to meet all of these requirements, we have designed Symbiote - a distributed Reconfigurable Logic Assisted multimedia Data Stream Management System (RLADSMS) at Indiana University Purdue University, Indianapolis (IUPUI) that provides hardware accelerated data stream processing using Field Programmable Gate Arrays (FPGAs).

## Categories and Subject Descriptors

H.2.6 [Information Technology and Systems]: [Datastream machines]; C.1.3 [Computer Systems Organization]: General—System architectures

## General Terms

Design, Performance

## Keywords

Data stream management systems, FPGAs, hardware accelerator

## 1. INTRODUCTION AND MOTIVATION

Research into Data Stream Management Systems (DSMSs) such as Aurora [1], Borealis [2], NiagraCQ [6], STREAM [3], Gigascope [8], Hancock [7], Telegraph [5], Tribeca [15], and Purdue Nile [11] amongst others have yielded elegant near real-time system architectures, query languages, and algorithms for processing scalar data streams generated from sensors/data sources reporting environmental conditions or statistical information such as temperature, pressure, traffic, or financial information.

However, traditional DSMSs make several assumptions that may not be ideal for multimedia DSMSs. First, traditional DSMSs assume “dumb” data sources (or sensors) that do not take part in

query processing. As a result, most (if not all) of the data stream processing is done on a central resource-heavy DSMS server. For future multimedia DSMSs, this assumption can have scalability implications such as (a) streaming multiple video streams to a central DSMS server can easily overwhelm its input bandwidth, and (b) the central DSMS server may not be able to cope with the computationally intensive task of performing complex image processing on multiple input video streams. Second, some multimedia monitoring systems have stringent real-time requirements that need to be met with minimal loss of information even in high-load scenarios. For example, in a traffic monitoring system, it is not only required that an accident be detected but also required that such an event be detected fast in order to minimize the overall response time for medical teams.

Moreover, streaming applications have several unique characteristics that make their hardware acceleration using custom processors more advantageous over traditional processors. First, data streams are unbounded in nature where tuples are often read only once before getting processed. Hence, the cache hierarchy of traditional processors optimized for data-reuse (i.e., temporal locality) does not offer any significant advantages. Additionally, traditional processors incur noticeable virtual-address translation overhead while accessing large data streams [13], often causing to miss real-time deadlines. On the other hand, as data streams are unbounded in nature, they need not be stored in the memory for a large amount of time. This is advantageous for hardware accelerators such as FPGA-accelerators as they are limited in terms of on-chip memory such as BlockRAMs and distributed BlockRAMs [17]. Second, traditional scalar processors are optimized for low-latency operations and do not exploit parallelism beyond what is exposed in the instruction window of the processor. However, as there is limited/no dependency between the tuples of the streams, hardware level data parallelism techniques such as SIMD processing on FPGA-accelerators become more attractive. Moreover, as streaming applications are computationally intensive with high computation to memory-access ratio, other FPGA-based techniques such as functional pipelining and VLIW microarchitectures also become favorable.

The aforementioned requirements of multimedia DSMSs and the advantages of hardware-based stream processing have motivated our research into Symbiote - one of the first demonstrative approaches that can perform image processing as well as various query operations in hardware using FPGAs. FPGAs are reconfigurable computing fabrics that can be configured to run parallel hardware implementations of algorithms. Hence, we have developed a hardware-software codesigned sensor node and central Symbiote (the name Symbiote represents the synergy between its hardware and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.  
Copyright 2010 ACM 978-1-4503-0032-2/10/06 ...\$10.00.

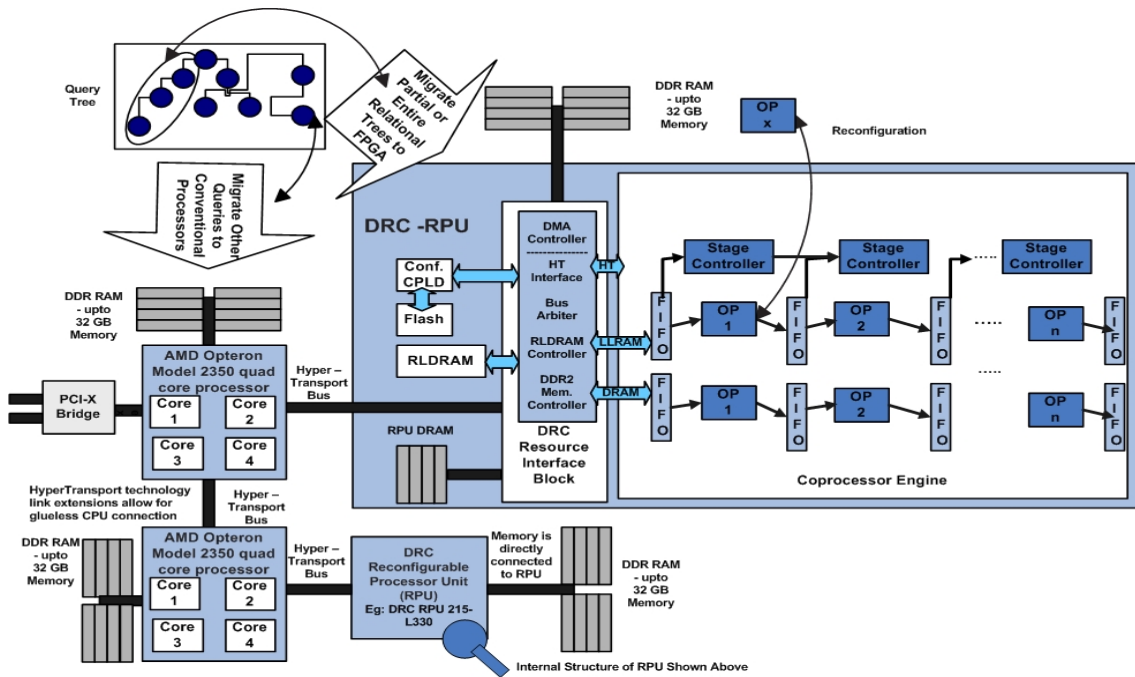


Figure 1: Architecture of Symbiote central DSMS server.

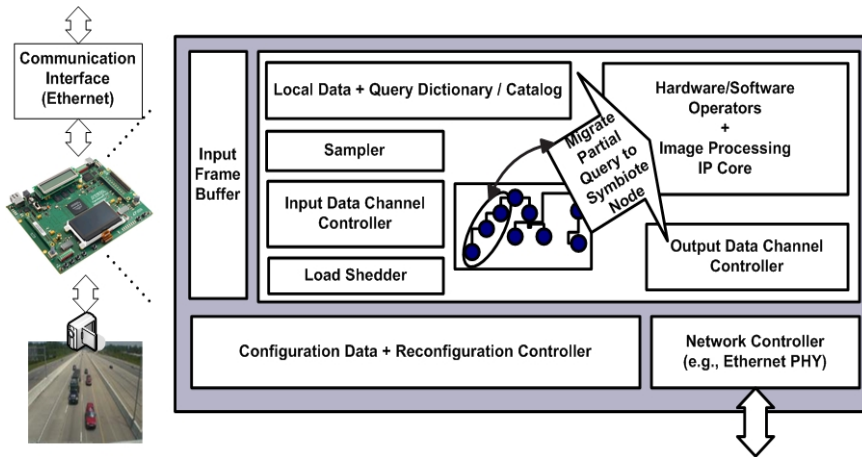


Figure 2: Symbiote sensor node architecture.

software functionality) DSMS server that together satisfy the real-time requirements and Quality-of-Service (QoS) needs of data intensive multimedia sensor network applications by leveraging the parallel execution capabilities of FPGAs with multicore processors. Our approach accelerates multimedia data stream computation on sensor nodes, minimizes the execution overhead on the central DSMS server, and maximizes the utilization of shared information bandwidth between sensor nodes and central DSMS server.

## 2. OVERVIEW OF SYMBIOTE

As shown in Figures 1 and 2, Symbiote is being designed as a two-tier hierarchical, distributed DSMS consisting of wired multimedia sensor nodes that acquire multimedia content (such as video) from the environment, locally process the multimedia content, and send relevant feature streams derived from the multimedia content up the hierarchy to the central Symbiote DSMS server. As the data flows from the Symbiote sensors to the Symbiote DSMS server, we

describe the Symbiote nodes first in Section 2.1 and then describe the Symbiote DSMS server in Section 2.2.

### 2.1 Symbiote Node Architecture

Figure 2 shows the architecture of the Symbiote node that is specifically designed for a real-time traffic monitoring application. Each node includes a video camera input, a Xilinx Virtex-4 FPGA, and is Ethernet capable. Moreover, each Symbiote node can directly communicate with the central DSMS server or with another Symbiote node in the vicinity according to the query plan generated by the central DSMS server in response to the query submitted by the human operator. When the query is submitted to the central Symbiote server, the query plan generator generates an optimized query plan and transmits control information to trigger the configuration of the Symbiote node with the query tree that can be executed on the FPGA or software (as shown in Figure 2).

Each Symbiote node has an input frame-buffer where the captured images are stored. These images are then read by the in-

put data controller, sampled by the sampler, and processed by the hardware image processing IP core that performs dynamic content-based image segmentation for vehicle tracking and traffic monitoring [4]. The image processing IP core can detect events such as lane changing, congestion, and accidents using information content associated with image sequences with high precision (the number of correctly detected cars / number of detected cars) and recall (the number of correctly detected cars / number of cars) rates.

As most of the image processing and datastream filtering operations are performed locally on the node itself, the computational overhead and bandwidth requirements on the central Symbiote server are minimized as compared to purely software approaches. Finally, as the intermediate scalar feature streams are generated by the Symbiote node, they are sent to the central Symbiote server using push-based asynchronous communication.

## 2.2 Symbiote Server Architecture

The central Symbiote server is designed to deal with a large number of scalar feature streams that are generated by the video sensor nodes. The hardware architecture of the central Symbiote server (shown in Figure 1) is based on a commodity DRC platform DS2004 [9] from DRC computer corporation. It contains two AMD Opteron 2350 quad core processors, 8 GB ECC DDR, an Nvidia 7300GT PCI Express video card, a 160GB SATA hard drive, and two Opteron socket-compatible DRC Reconfigurable Processor Units (RPU). The RPUs each contains a Virtex-5 FPGA that can be configured to execute data stream operators natively in hardware. The RPUs are also tightly coupled with direct access to DDR memory and any adjacent Opteron processor at HyperTransport (HT) [12] bandwidth and low latency. The CPU to CPU bandwidth is at full HT 3.0 bandwidth of 21 GB/sec [12] while CPU to RPU bandwidth is 4 GB/sec. This system is capable of hosting ccNUMA operating system namely Linux (64 bit) Ubuntu 8.04. The software infrastructure of Symbiote is derived from the Borealis DSMS [2]. However, we have extended it in several non-trivial ways to provide accelerated hardware-software co-execution of data stream operators as described in the following sections.

### 2.2.1 Hybrid Data Model

Most DSMSs use a row-oriented storage model for representing tuples from a stream. A stream ‘S’ is defined as an unbounded, append-only multiset of tuples  $(t_s, v_1, \dots, v_n)$ , where  $t_s$  is the time stamp that denotes logical arrival time of a tuple on stream ‘S’ and  $v_1, \dots, v_n$  are values of attributes defined on a data schema such as  $(T_S, A_1, \dots, A_n)$ .

However, as most DSMS queries accept and process entire tuples but refer to only specific fields to perform filtering and correlation of information, alternative storage models such as the decomposed storage model recently employed by database systems may provide benefits for DSMS queries. In the decomposed storage model, an ‘n’-ary relation is split into ‘n’ binary relations called Binary Association Tables (BATs). Each BAT consists of two attributes namely, an object ID (OID) attribute and a large contiguous array of actual values for the ‘n<sup>th</sup>’ attribute. It is the authors’ contention that such BATs of the column-oriented storage model are similar in structure to the densely packed matrices/vectors commonly found in high performance computing. Hence, vectorized matrix operations [10, 14] performed using hardware accelerators for scientific computations may find parallels in accelerating data streams. The intuition behind this reasoning is further supported by the very nature of DSMS. First, data streams are read-only and in most cases DSMS queries access only specific attributes. Hence, the decomposed storage model can be more IO-friendly as only the stream

attributes required for the data stream queries can be moved closer to the hardware accelerators. Second, as most data stream operators have array-loop intensive code patterns, each database operator can be mapped into SIMD computation on hardware accelerators.

The main issue in using decomposed storage models for data stream systems is that the decomposed storage model may not be semantically ideal in the context of data stream systems where data stream sources repeatedly capture a set of attributes and then stream them as row-oriented data tuples. The challenge is to maintain a balance between the way tuples are produced by the data sources and the requirement of decomposed storage model for hardware accelerators. Hence, the Symbiote DSMS server uses a hybrid data model for storing tuples. It uses a traditional row-oriented storage model for data streams processed by software query operators that run on traditional processors whereas the decomposed storage model is used to store data streams processed by hardware query operators. The decomposed storage model is essential so that (a) the hardware query operators can be reused across various data types supported by Symbiote, and (b) the hardware query operators could deal with various tuple sizes and schemas. In order to facilitate conversion between the hardware and software data storage domains, we have designed two novel query operators called split-choose and merge. These operators are transparently inserted into a query network between the software and hardware execution domains when a query is executed and/or optimized. The split-choose operator splits the stream into columns, selects columns that are required by the hardware query operators, and then streams the columns to the hardware query engines. On the other hand, the merge operator merges the query results produced by the hardware query engines into the row-oriented tuple storage data model.

### 2.2.2 Query Operators

As Symbiote is derived from Borealis, it provides six primitive operators namely filter, map, union, bsort, windowed join and windowed aggregate operators in software. The windowed software operators provide optional timeout and slack parameters that enable the Symbiote DSMS to deal with bounded disorder in the form of slow or out-of-order arrivals of data tuples, respectively. Additionally, we have developed corresponding parameterizable hardware implementations of each of these operators. For example, we have developed a parallel hardware version of the join operator similar to our hardware join operator for column-oriented databases [16]. This operator performs at least an order of magnitude faster than the software data stream operator.

### 2.2.3 Query Language Interface

We have extended the Borealis GUI [2] to provide two distinct views of the Symbiote architecture. First, the “sensor-view” is used to specify the association between the sensor nodes and the data sources as well as the query tree executed natively in hardware associated with the sensor nodes. The GUI infers the properties of the sensor node interconnects (such as interconnect latency, bandwidth, and QoS parameters for each of the data transfer arcs) based on the knowledge of network interconnects deployed in field. Additionally, depending on the sensor nodes involved in the query, the GUI will infer properties such as whether the given node is a video sensor node, camera properties, frame rate, hardware resource usage, embedded memory usage as well as on-chip and off-chip memory available on the sensor node. These properties are inferred and stored in the central DSMS server so that they can be used by the query optimizer described in Section 2.2.4. Second, the “query-view” is similar to the stream viewer designed in Borealis [2] and is used to specify the entire procedural query language.

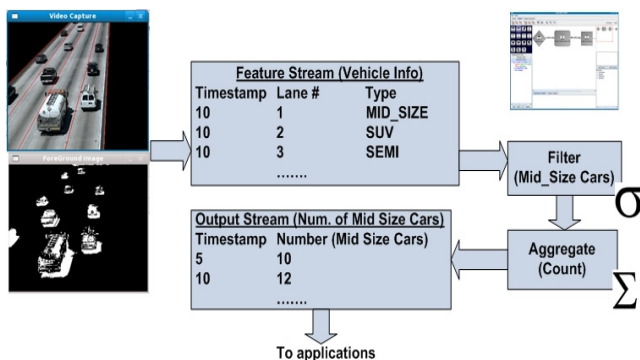


Figure 3: Sample demonstration application.

### 2.2.4 Symbiote Run-time

The key components of the Symbiote run-time are the hybrid storage manager, the scheduler, the query optimizer, and load shedder. The hybrid storage manager is designed to store and manage the input queues of tuples. This storage manager is responsible for storing the tuple representations in row-oriented or decomposed storage format depending on whether the streams are processed by software or hardware operators. In conjunction with scheduler, this storage manager is also responsible for moving the data to (or off) the hardware memory from (or into) software accessible memory.

The scheduler is responsible for deciding which operators execute in software or hardware. The scheduler also transforms a partial query tree executed in hardware to corresponding control messages that invoke the execution of the hardware query operators (as shown in Figure 1). Moreover, it is responsible for determining the boundaries of hardware-software coexecution by inserting the split-choose and merge operators described in Section 2.2.1.

The query optimizer performs static optimization and decides whether certain operators execute in hardware versus software. The optimization algorithm tries to move I/O intensive boxes near to the sources that are producing the data streams. If the boxes are hardware only operators (such as the image processing IP core), it will first allocate spatial area to such boxes. It will then configure the hardware to execute hardware data stream operators. Each of these operators is designed to provide a minimum run-time complexity. Then moving up the data-flow network, the algorithm merges groups of boxes into superboxes that can be synthesized to hardware. This entails comparing the run-time complexity of the hardware operators with that of the software operators. Furthermore, additional decisions such as co-locating boxes with tables (similar to Borealis) are performed in this step. We are still investigating how the load-shedding can be done in hardware. However, in high-load scenarios Symbiote relies on a software-based load shedder similar to [2] to drop tuples from input data streams.

## 3. SYSTEM DEMONSTRATION

We demonstrate a sample traffic monitoring system consisting of two Symbiote sensor nodes that monitor two different video streams. The image processing IP cores then detect the lanes and the types of vehicles in each lane in order to generate the feature streams that are filtered locally and then forwarded to the Symbiote server.

Users can interact with the system by deploying queries using the GUI described in Section 2.2.3. The GUI lets users visualize the distributed architecture of Symbiote, optimization strategy, and the co-execution of hardware/software query operators. Additionally, users can view the processed data streams and observe the

reduction in the latency of the tuples delivered to the applications by comparing the execution of queries in hardware with the execution of queries in software. We also demonstrate sample queries such as “how many mid-size cars have passed through the monitored road?” and “does toll-collected at the toll-station match the toll that should be collected (based on the size of the vehicles)?”. Figure 3 shows one such query where the Symbiote node detects the lanes and the types of the vehicles traveling on the road in order to generate a *Vehicle Info* feature stream. This feature stream is then filtered for “mid-size” cars, and the aggregate (count) number of mid-size cars traveling on the road is generated every 5 minutes. Additionally, we show how the query can be run in hardware or software using the Symbiote DSMS server.

## 4. ACKNOWLEDGMENTS

This research is being supported by NSF grant 0834682.

## 5. REFERENCES

- [1] D. Abadi et al. Aurora: A data stream management system. In *SIGMOD*, pages 666–666. ACM, 2003.
- [2] D. Abadi et al. The design of the Borealis stream processing engine. In *CIDR*, pages 277–289, 2005.
- [3] A. Arasu et al. Stream: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26:665–665, 2003.
- [4] F. Bowen et al. Dynamic content based vehicle tracking and traffic monitoring system. In *SPIE Electronic Imaging*, volume 6497.
- [5] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing. In *SIGMOD*, pages 668–668, 2003.
- [6] J. Chen, D. DeWitt, J. Tian, Y. Feng, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *SIGMOD*, pages 379–390. ACM, 2000.
- [7] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock: A language for analyzing transactional data streams. *TOPLAS*, 26:301–338, 2004.
- [8] C. Cranor, G. Yuan, T. Johnson, V. Shkapenyuk, and O. Spatscheck. Gigascope: High performance network monitoring with an SQL interface. In *SIGMOD*, pages 623–623. ACM, 2002.
- [9] DS2004, 2009. DRC Comp. Corp. [online] [http://www.drccomputer.com/pdfs/DRC\\_DS2000\\_fall07.pdf](http://www.drccomputer.com/pdfs/DRC_DS2000_fall07.pdf).
- [10] M. Gokhale, A. Jacob, C. Ulmer, and R. Pearce. Hardware technologies for high-performance data-intensive computing. *Computer*, 41(4):60–68, 2008.
- [11] M. Hammad et al. Nile: A query processing engine for data streams. In *ICDE*, pages 851–851, 2004.
- [12] HyperTransport, 2010. HyperTransport Consortium, <http://www.hypertransport.org/>.
- [13] S. Rixner. *Stream processor architecture*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
- [14] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Efficient vectorization of the FIR filter. In *Annual Workshop on Circuits, Systems and Signal Processing*, pages 432–437, Nov. 2005.
- [15] M. Sullivan. Tribeca: A stream database manager for network traffic analysis. In *VLDB*, page 594, 1996.
- [16] P. Vaidya and J. Lee. A novel multicontext coarse-grained join accelerator for column-oriented databases. In *ERSA*, pages 158–164, 2009.
- [17] Virtex5, 2010. Virtex-5 FPGA Family, <http://www.xilinx.com/products/virtex5/>.