On Network CoProcessors for Scalable, Predictable Media Services

Raj Krishnamurthy, *Student Member*, *IEEE*, Karsten Schwan, *Senior Member*, *IEEE*, Richard West, and Marcel-Catalin Rosu

Abstract—This paper presents the embedded realization and experimental evaluation of a media stream scheduler on Network Interface (NI) CoProcessor boards. When using media frames as scheduling units, the scheduler is able to operate in real-time on streams traversing the CoProcessor, resulting in its ability to stream video to remote clients at real-time rates. The contributions of this paper are its detailed evaluation of the effects of placing application or kernel-level functionality, like packet scheduling on NIs, rather than the host machines to which they are attached. The main benefits of such placement are 1) that traffic is eliminated from the host bus and memory subsystem, thereby allowing increased host CPU utilization for other tasks, and 2) that NI-based scheduling is immune to host-CPU loading, unlike host-based media schedulers that are easily affected even by transient load conditions. An outcome of this work is a proposed cluster architecture for building scalable media servers by distributing schedulers and media stream producers across the multiple NIs used by a single server and by clustering a number of such servers using commodity network hardware and software.

Index Terms—Cluster machines, multimedia services, embedded systems, quality of service, operating systems, real-time systems, data streaming, packet scheduling.

1 Introduction

1.1 Background

The scalable delivery of media and Web services to end users is a well-recognized problem. At the network level, researchers have developed multicast techniques [4], media caching or proxy servers [2], reservation-based communication services [16], and specialized media transmission protocols [25]. For server hardware, scalability is sought by using extensible SMP and cluster machines [5], [11]. Scalability for server software is attained by using dynamic load balancing across parallel/distributed server resources [50], and by using admission control and online request scheduling [51], [66] for CPUs [41], [9], [24], network links [63], [62], [66], and disks [11]. Complementary to such work are application-level or end-to-end solutions [35], [46] that adapt server and/or client behavior in response to changes in users' Quality of Service (QoS) needs and in resource availability [59], [47].

1.2 Scalable Cluster Services

This paper addresses the scalability of media servers. Its approach utilizes servers constructed as clusters of processor/storage nodes, each of which has a network interface processor (NI) linking it to the cluster's high performance system area network [60]. A unique aspect of the approach is that each NI has a programmable CoProcessor with local memory, direct access to devices, and the ability to perform both basic protocol processing and certain application-specific tasks. The CoProcessors explored to date include ATM FORE [49], Myrinet [60], I2O-compliant network interface boards [22], [38], [21], and, most recently, gigabit Ethernet attached to IXP1200 [55], [52], [18], [68] boards. We have also explored Xilinx FPGA CoProcessor boards [65] in servers with gigabit links and the ability of the FPGA CoProcessor to meet the packet-time requirements of 10Gbps links [30]. The server hardware configuration used in this paper is from a generation of CoProcessors (developed for the I2O standard) equipped with the i960RD CoProcessor (our systems software does not use any I2O standard-style driver partitioning or any of the hardware registers in the I2O hardware units). This research simply uses the i960RD processor on the NI with associated PCI bridge chipsets, as a i960RD CoProcessor. These NIs have two 100Mbps Ethernet links, a PCI interface to the host CPU and two SCSI interfaces directly attached to disk devices. These CoProcessors are attached to a prototypical server system comprised of 16 quad Pentium Pro nodes, where host-to-host communications are supported by VxWorks TCP/IP stack protocols (like TCP and UDP) and where media streams may flow from server disks to clients via host CPUs or directly via the i960RD boards [38], [21].

The approach to server organization we advocate is one that views a server like the one depicted in Fig. 1, as an information processing, storage, and delivery engine that is programmed at two levels of abstraction, reflecting the hardware configuration being employed:

1. Low-level services execute on the CoProcessors (NIs), as demonstrated in Fig. 1, for the concrete

- *R. Krishnamurthy and K. Schwan are with the Center for Experimental Research in Computer Systems, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: {rk, schwan}@cc.gatech.edu.*
- *R. West is with the Department of Computer Science, Boston University, 111 Cummington Street, Boston, MA 02215. E-mail: richwest@cs.bu.edu.*
- *M.-C. Rosu is with the IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598. E-mail: mrosu@watson.ibm.com.*

Manuscript received 13 Nov. 2001; revised 24 Oct. 2002; accepted 14 Jan. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 115364.

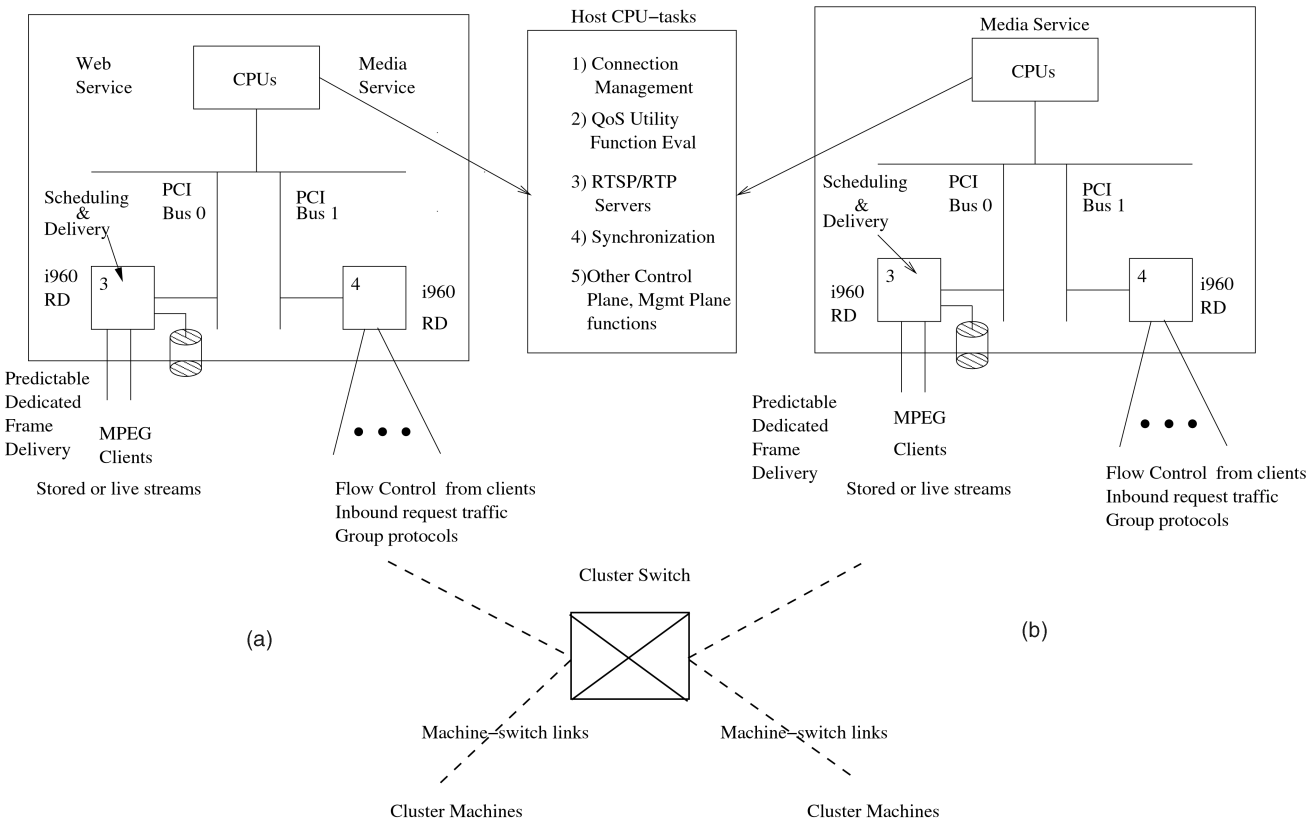


Fig. 1. Cluster hardware: Host CPU, NI CoProcessor, and interconnect. (a) Multiservice servers and (b) single-service servers.

examples of frame scheduling and predictable frame delivery for streaming media services. Whether executed synchronously or asynchronously with application programs' communications, these services may be viewed as application-specific *extensions* supported by a *runtime* resident on the NI.

2. Higher level services run on host nodes, but they may utilize CoProcessor services like media scheduling via an explicit NI-provided interface. Fig. 1 shows sample media service tasks that are run on host CPUs, with predictable frame delivery dedicated to run on NIs.

1.3 Contributions

Our approach to improving the scalability of media servers is to amplify their capabilities by extension of their communication CoProcessors with application-specific functionality. The intent is to use cluster nodes with their complete OS functionality, large memories, and deep cache hierarchies for the computational and data management tasks for which they are well suited, while using NIs for communication-centric tasks like packet scheduling. The following insights are gained by this experimental research:

1. *Efficient execution on standard NIs.* Even NIs like Intel's i960RD-based boards [38], [21] are feasible platforms on which to run application-specific extensions of communication functionality. Specifically, performance-critical communication extensions can be executed with high performance on such COTS (Commercial Off-The-Shelf) CoProcessors running with standard operating system software (e.g., VxWorks [64]). This is encouraging, given the fast-moving nature of NI hardware development and the onerous task of porting and reporting the specialized runtime software used in previous work [45], [17].
2. *Improved predictability for NI-based streaming services.* On host CPUs, the time-critical execution of services is easily jeopardized by the need to run a mix of applications. Specifically, streaming services like media schedulers running on host-CPUs are easily affected even by transient loading conditions, whereas media schedulers running directly on NIs are immune to such host-CPU loading. The key result is that CoProcessor-based scheduling substantially improves the jitter experienced by streaming services. Such improvements are particularly important for real-time media services like remote surveillance or telepresence [56], [33].
3. *Improved server scalability by use of extended NIs.* Services that frequently execute *device* or *network-near* functions are known to run faster on CoProcessors because their execution does not involve I/O busses, host memory, and host CPUs. This paper presents results that better quantify the load reductions—termed *traffic elimination*—on such node resources derived from NI-based service execution, thus freeing up these resources for use by other server tasks. Traffic elimination is attained by implementing stream-selective lossiness in overload conditions via window and time-constrained scheduling of MPEG video frames, employing the DWCS

(Dynamic Window Constrained Scheduling) algorithm [63], [62]. Packet scheduling also serves to guarantee differential packet rates and deadlines to meet clients' individual QoS needs and to eliminate traffic.

1.4 Summary of Experimental Results

A DWCS-based media scheduler can run almost as fast on a relatively slow CoProcessor as its corresponding implementation on a standard workstation platform. Specifically, the scheduling latency of the host-based implementation of DWCS reported in [63], [62] is $\approx 50\mu$s on an Ultra Sparc CPU (300 MHz) with quiescent load. In comparison, the scheduler's execution on a 66 MHz i960RD CoProcessor is $\approx 67\mu$s, despite the fact that the i960RD is roughly four times slower than the Ultra Sparc host CPU. Reasons for high scheduler performance on the CoProcessor are presented in Section 3.2.

Performing packet scheduling on the NI rather than the host reduces load on the server's I/O busses, CPU, and memory resources. In one experiment described in Section 3.3, media content are sent directly from the NI to a remote host, thus eliminating $\approx$ 132 MB/sec of data load from PCI bus and memory of the host node for a 32-bit wide bus running at 33 MHz. With multiple media streams, the load eliminated consumes a substantial fraction of the machine's available PCI bandwidth. The importance of reducing PCI load is evident when considering that a media stream that fully utilizes 1Gbps links would consume 25 percent of the capacity of a 64bit/66MHz PCI interface, which shows that multigigabit interconnects (e.g., 10Gbps Ethernet) would consume more than the capacity of the fastest currently available 64bit/66MHz PCI interfaces.

Section 3.4 demonstrates the comparatively high predictability of CoProcessor versus host-based media services, by showing that transient loads can substantially affect host-CPU based schedulers, observing performance degradation for media scheduling even for relatively low CPU utilizations (i.e., 45 percent) and severe degradation for high CPU utilizations (i.e., over 60 percent).

1.5 Previous Work

Earlier work conducted by our group did not address scalable cluster media services. Further, rather than using standard host-CoProcessor interfaces, we developed and experimented with zero-copy interfaces and with a software architecture for realizing a rich set of communication instructions on NIs, termed a *(Distributed) Virtual Communication Machine* (DVCM [45]). The idea was to permit application programs to dynamically extend the current set of communication instructions resident on the NI to support their specific needs [45], [48], as subsequently also done in the SPINE project [17]. As with SPINE and other prior work on dynamically extending operating system kernels [14], [8], the services implemented by the DVCM can vary over time, in keeping with the needs of current cluster applications. The prototype described in this paper does not reimplement all of DVCM. Instead, our goals are 1) to gain an understanding of how to build a scalable media server from clustered node-NI pairs, 2) to evaluate, in detail, an NI's ability to support multimedia scheduling services, and 3) to compare NI versus host-based scheduling service realizations. Furthermore, while an earlier publication by our group already demonstrated the basic ability of a CoProcessor to perform media scheduling [29], this paper provides basic insights into the benefits of using CoProcessors or hosts, including an analysis of the overheads and trade offs concerning media scheduling with respect to locating media schedulers on CoProcessors versus hosts, using multiple versus single CoProcessors. In addition, we evaluate the utility of certain CoProcessor hardware, such as the benefits derived from caches, software floating-point, and specialized scheduler hardware units.

2 Network CoProcessor-Based Media Scheduling

2.1 Software Architecture of NI-Based Application Services

Our NI-based support for application-specific services is structured as three sets of software modules: host interface, runtime support, and application-specific extensions.

2.1.1 Host Interface

The interface functions exported by the NI to the host make NI-resident communication services appear to application programs as specialized "communication instructions." These instructions are accessible via memory-mapped pages shared by a host-resident application and the NI-resident media scheduling service. Pages contain control information as well as the communication buffers used for message transfers from NI to host and vice versa, much like the efficient message interfaces used in high performance messaging systems like FM [40].

With this interface, media frame producers running as application threads may stream frames to remote clients using the NI-resident scheduler. Remote consumers may forward frames to other consumers or buffer frames for display or storage, where frames are scheduled for delivery by the NI-resident scheduler.

2.1.2 Runtime

This set of NI-based modules supports the implementation of application-specific NI functionality. Using the VxWorks real-time operating system [64] as a basis, additional functionality to exploit this NI's specific hardware and to implement efficient media scheduling include a fixed-point library for efficient implementation of certain scheduling computations, driver front-ends to initialize controllers/storage, timestamp counter rollover management, and circular queues and heaps as the buffer structures used for media frames.

2.1.3 Extensions

NI extensions support specific applications' needs. Examples evaluated in our previous work include atomic read and write operations for remote NIs and efficient implementations of cluster-wide synchronization operations [45], [48]. This paper implements and evaluates the DWCS scheduler for streaming media on the i960RD cards.

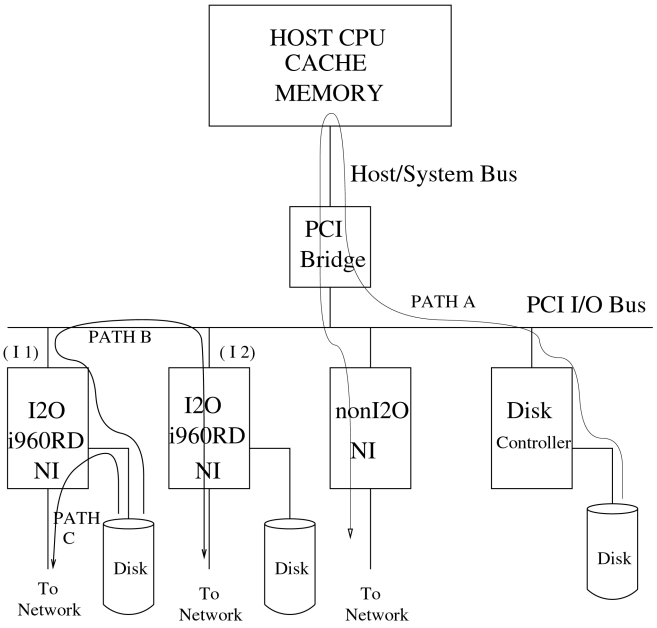


Fig. 2. Frame transfer paths.

2.2 A CoProcessor-Based Media Scheduling Service

2.2.1 Alternative Service Configurations and Basic Performance Factors

The performance of a media scheduling service is strongly affected by the distribution of streams and schedulers across the underlying cluster node/NI pairs. Alternative stream and scheduler configurations are depicted in Fig. 2, where multiple NIs (in this case, i960RD cards) are present on each cluster node's I/O bus. One or more of these NIs may run the media scheduler and also support disks directly attached to them. This results in three possible paths traversed by the media frames being scheduled, shown in Fig. 2, as Paths A, B, and C.

Path A represents the case where media scheduling is performed only on host CPUs, which also implies that all media streams must touch upon hosts. As a result, frames sent to clients by the server are transferred from a server disk attached to an SCSI controller card, to server host memory (via a PCI interconnect), then again transferred via the PCI interconnect to a non-i960RD NI and, finally, sent via the network to whichever clients requested the media stream. Stated in more detail, the server's node OS transfers the MPEG file from disk to its filesystem buffer cache, then to the application-level thread that has opened the MPEG file for reading. This involves at least two memory copies as well as the traversal of memory hierarchies and of bus domains (i.e., I/O bus to system bus). After having been scheduled, the second part of Path A involves the transfer of frames from host CPU memory to the network via the NI, again involving multiple memory hierarchy and bus traversals.

Path B depicts a configuration in which a media stream originates either on another machine or on disks directly attached to an i960RD card (I1) and is then sent to some client via a second i960RD card (I2) running the media scheduler. This path represents the general case of NI-based media stream scheduling, involving multiple NIs attached to one server node, with each NI specialized to perform certain tasks. This path involves the I/O bus, but completely eliminates the uses of host CPU or memory.

Finally, the "best" case in terms of host node resource usage is depicted in Path C, where a single NI acts as both the source of the media stream using a disk attached to it and also executes the media scheduler. Compared to Path A, this path eliminates uses of the I/O bus, host CPU, and memory. While imposing additional load onto the NI, one potential advantage of this configuration is the relative "closeness" of the media scheduler to the network, which facilitates rapid configuration in response to changes in network behavior. A sample dynamic reconfiguration is one that adjusts the scheduler's degree of lossiness in response to observing changes in the number of packet retransmissions currently experienced for this media stream. Another possible advantage of this configuration is the ability to share a single set of buffers between the NI's disk and network interface, thereby avoiding additional memory copies [39].

Summarizing the performance characteristics of different service configurations:

1. Performance is impacted both by the speed of scheduling actions and by the total percentage of a media stream's data that traverses a cluster node's memory hierarchies.
2. It is affected by the total protocol processing overheads experienced on NI and/or nodes and by the disk access overheads experienced for media streams originating at or destined to storage devices.
3. It also depends on the number of bus-domain traversals (system bus to PCI bus and vice versa) experienced by streams.
4. Scheduling and streaming performance are also affected by the presence of other programs executing on nodes and NIs, measurable not only as changes in throughput, but also in terms of the delay-jitter and loss experienced for media streams and scheduling actions.

2.2.2 Host-Based Scheduler Implementation

Our media scheduler is based on the DWCS (Dynamic Window-Constrained Scheduling) algorithm described and evaluated in [63], [62]. While shown to be a highly efficient algorithm, most important to this paper are the ways in which the performance of this scheduling service is affected by alternative implementation choices for its host versus NI-based realizations. Factors governing performance include the respective hardware and operating system platforms, certain hardware features, and memory structures. This large variety of factors precludes analytical comparisons of scheduling performance based on the alternative paths shown in Fig. 2. For instance, the media scheduler's implementation on the host CPU running SUN's Solaris operating system is embedded into a separate scheduling process. By presenting a shared memory-based API (using System V shared memory) to processes that generate media content, this media scheduler is made independent of the source of media data (e.g., from host-attached disks or from remote nodes), many media streams can be scheduled by a

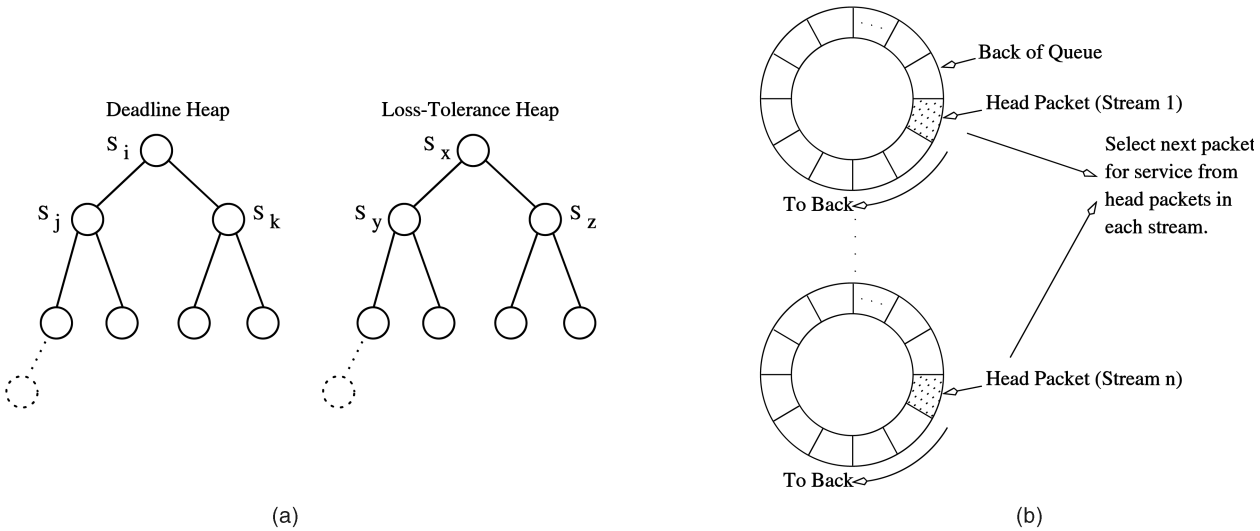


Fig. 3. (a) The NI implementation of DWCS uses two heaps: one for deadlines and another for loss-tolerances. (b) Using a circular queue for each stream eliminates the need for synchronization between the scheduler that selects the next packet for service and the server that queues packets to be scheduled.

single host-resident scheduler, and packets in any stream may be enqueued in scheduler data structures, concurrent with scheduling analysis and dispatch of previously enqueued packets. Additional scalability in scheduler operation with respect to number of packet streams, stream rates, and "tightness" of deadlines and loss-tolerance is achieved by varying the rates of scheduler actions [63], [62].

2.2.3 The NI-Based Scheduler Implementation

The comparatively lighter-weight NI-embedded implementation of the DWCS scheduler is shown in Fig. 3. It is layered on top of the NI's VxWorks real-time operating system. It runs as a single VxWorks thread, uses pinned memory for disk and/or network buffers and to interact with other NI-resident threads. Its host interface maps some of its memory to the host via the PCI device. Compact data structures (scheduler attributes or scheduler frame descriptors) for packet schedule representation minimize the use of NI memory and memory usage is reduced further by not copying frames, whenever possible. These implementation choices attempt to compensate for the relative resource paucity on the NI. In addition, the NI-based scheduler takes advantage of certain hardware features existing on the i960RD card. First, the code used for scheduling analysis is decoupled from the schedule representations (i.e., scheduling data structures). The intent is to evaluate the performance effects of using alternative representations, including those that use the i960RD's hardware-supported FCFS circular buffer queues. This is important because, with DWCS scheduling, packets in a given stream (at the same priority level) may be scheduled based on a service tag associated with each packet. Second, by using one thread for both packet scheduling and dispatch, a single data structure can hold all frame descriptors (or the attributes describing them), thus conserving memory. Also, packets will not experience additional queuing delay and jitter in dispatch queues [63], [62].

The NI-based scheduler operates as follows: It is booted in conjunction with the VxWorks Operating System, from flash-ROM on the i960RD NI card. Initialization code in the kernel spawns the scheduler thread. Any media frames received by the NI-based scheduler are temporarily stored on the NI, using the i960RD NI's 4MB of on-board memory, which may expanded to 36MB. To conserve memory, only a single copy of "to-be-scheduled" frames is kept in NI memory and scheduling analysis and dispatch directly manipulate addresses of frames. Frames are stored on a per-stream basis. Head-of-line packets in each stream form loss-tolerance and deadline heaps and encode stream priority values. The scheduler must pick the stream with the highest priority according to rules described in [63], [62].

Storing frames directly in NI memory (rather than in host memory) reduces the overall scheduling analysis and dispatch latency for each frame and the jitter experienced by a sequence of frames. It also reduces mean frame queuing delay since frames need not be "pulled" from host memory. A detailed analysis of the performance effects of locating various data structures on NIs versus hosts appears elsewhere [45].

As shown in Fig. 3, a circular buffer is maintained for each stream, with separate head and tail pointers. Frame producers inject frames into the scheduler using the tail pointer and the scheduler reads frames using the head pointer. This eliminates any explicit synchronization needs between readers and writers. A deeper understanding of the scheduler's operation requires knowledge of the algorithm it executes, which is outlined next.

2.2.4 Some Details about the DWCS Scheduling Algorithm

DWCS is designed to maximize network bandwidth usage in the presence of multiple packets that have individual delay constraints and loss-tolerances. The per-packet delay and loss-tolerance attributes are derived from higher-level application constraints:

- *Deadline*—This is the latest time a packet can *commence* service. It is determined from a specification of the maximum allowable time between servicing consecutive packets in the same stream.

- *Loss-tolerance*—This is specified as a value x_i/y_i, where x_i is the number of packets that can be lost or transmitted late for every *window*, y_i, of consecutive packet arrivals in the same stream, i. For every y_i packet arrivals in stream i, a minimum of $y_i - x_i$ packets must be scheduled on time, while at most x_i packets can miss their deadlines and be either dropped or transmitted late, depending on whether or not the attribute-based QoS for the stream allows some packets to be lost.

At any time, all packets in the same stream have the same loss-tolerance, while successive packets in the same stream have deadlines that are offset by fixed amounts from their predecessors.

Using these attributes, DWCS has the following abilities:

1. It can limit the number of late packets over finite numbers of consecutive packets in loss-tolerant or delay-constrained, heterogeneous traffic streams.
2. It does not require a priori knowledge of the worst-case loading from multiple streams to establish the bandwidth allocations necessary to meet per-stream delay and loss-constraints.
3. It can safely drop late packets in lossy streams without unnecessarily transmitting them, thereby avoiding needless bandwidth consumption.
4. It exhibits both fairness and unfairness properties when necessary.

Proofs of these properties and additional detail about DWCS appear in [63], [62]. West and Poellabauer [61] describe an approach where deadlines are compared before window-constraints. This approach was intended to provide hard guarantees on loss constraints. The slight variations of the DWCS algorithm in [61] currently being investigated do not concern its running time or performance and are therefore not relevant to this paper's contents, which is based on the original version of DWCS [63], [62].

2.2.5 Discussion

One property of DWCS (and of other media schedulers) is the fact that packets are dropped under certain conditions. This means that a scheduler operating on the NI will typically not forward all packets to the host and/or to other recipients. The resulting reduction in node resource usage is one motivation for placing scheduling onto the NI. Another motivation is the network-near nature of an NI-resident media scheduler, which enables it to rapidly change its operation in response to changes in network conditions. In previous work, we have demonstrated performance advantages due to such network nearness for other NI-resident communication services [48]. Furthermore, in previous and ongoing research, we are demonstrating the advantages derived from traffic filtering when placing computations other than media scheduling onto NIs, such as the downsampling of media of scientific data [42], [23], both of which use application-level header information to eliminate certain packets from a data stream. Other methods for data stream downsampling, however, like image or sensor data conversion [67], require levels of processing power not offered by NIs like the i960RD boards used in this paper. They require additional processing power, for which we are experimenting 1) with FPGA devices attached to NIs [26], [27], [30] and 2) with network processing microengines on IXP boards [68].

3 Insights and Experimental Evaluation

In addition to validating the feasibility of DWCS-based media scheduling on NIs, this section also demonstrates performance advantages derived from this approach.

3.1 Experimental Setup

Experiments use a typical PC-based server platform, in our case a Quad Pentium Pro server (4 X 200MHz CPUs) running Solaris 2.5.1 X86 with 128 MB of memory. Three NI cards are placed into separate PCI slots on the same bus segment. One NI hosts the scheduler and scheduler data structures. The other two cards serve as stream sources for MPEG traffic. Disks used as stream sources are directly attached to the NIs and to hosts. An MPEG segmentation program [63], [62] is used to partition an MPEG-encoded file into I, P, and B frames, thereby emulating the MPEG file segmentation process in an MPEG player. The MPEG segmentation process may be run on the host CPUs or the NIs. MPEG stream producers on the host or NIs inject frames into the scheduler queues on the scheduler NI using PCI bus transfers. The scheduler picks frames based on scheduling criteria and dispatches them to the network. Client machines running MPEG players may attach to the scheduler card for MPEG stream delivery. Built-in monitoring mechanisms measure desired performance parameters at the scheduler card or at the remote client end. Remote client machines connect to the scheduler NI using a 100Mbps Ethernet switched interconnect [63], [62], [21], [38], [64].

3.2 Scheduling on NIs: Basic Capabilities

3.2.1 Microbenchmark Definition

Microbenchmarks measure the basic performance of alternative scheduler implementations and placements. In all such benchmarks, the scheduler is started only after all frames have been written into its circular buffer, which then contains the addresses of frame descriptors. "Total Sched Time" is the time to schedule all of the frames denoted by these addresses and to dispatch them to the network. "Avg Frame Sched Time" is the average time to schedule a *single* frame on the network. "Total time w/o Scheduler" is the cumulative time to transmit all frames on the network without the scheduler. These measurements are attained by rerouting execution in the code to a point where the address of the frame to be dispatched is readily available and does not need scheduler rules. "Avg Frame Time w/o Scheduler" is the average time to transmit a single frame without the scheduler.

3.2.2 NI-Based Scheduling is Feasible

Scheduling overhead on the i960RD NIs is experimentally shown to be $\approx 67\mu$s. This compares favorably with the scheduling latency of the host-based scheduler reported in [63], [62] as $\approx 50\mu$s. It also corresponds to about half an Ethernet frame time ($\approx 120\mu$s) on a 100Mbps link, thus indicating that it is viable to schedule multipacket MPEG frames at network speeds. It also suggests, however, that finer-grain (e.g., per packet) scheduling would consume too much of the NI's CPU cycles and, thus, require processing resources beyond those available on this and, likely, on other NIs.

TABLE 1
Scheduler Microbenchmarks (Data Cache Effects)

Microbenchmark	Software FP (μSecs)		Fixed Point (μSecs)	
	Cache Disabled	Cache Enabled	Cache Disabled	Cache Enabled
Total Sched time	19580.88	17398.56	16425.36	14295.60
Avg frame Sched time	129.67	115.20	108.48	94.60
Total time w/o Scheduler	5210.88	4776.48	4583.28	4195.68
Avg frame time w/o Scheduler	34.6	31.40	30.35	27.78

3.2.3 NI-Based Scheduling Requires Tuning

Table 1 records microbenchmarks for both a software floating-point version and a fixed-point version of the DWCS scheduler. The table also shows that efficient scheduler operation on the relatively slower NIs requires some degree of tuning. For example, floating-point computations are used in loss ratio computations performed in DWCS. However, like other NIs, the i960RD does not have a floating-point unit. Wind River Systems (see [64]) has provided a software floating-point (FP) library that may be configured into the VxWorks kernel. Measurements depicted in Table 1 show that software floating-point computation results in undue scheduling overheads. We address this issue by development of a DWCS-specific fixed-point version of the library, where arguments are simply stored as fractions with numerator and denominator, with divisions implemented as shifts. This reduces computation latency by 20μs, resulting in a latency of $\approx 78\mu$s for a DWCS scheduling decision (i.e., the difference between "Average frame Sched time with scheduler" and "Average frame Sched time without the scheduler") for the case of fixed-point computation. The quality of scheduling, expressed in terms of parameters like delay-jitter, loss, and throughput, is not affected because the explicit representation and manipulation of numerators and denominators have the numerical accuracy required by the scheduler's operation.

Additional reductions in scheduling latency on the NI demand that the data cache on-chip be enabled.[1] Table 1 shows that the presence of the data cache improves average frame scheduling time for both the software FP and the fixed point implementations of DWCS by $\approx 11\mu$s. These improvements are due to the caching of scheduler data structures, specifically, of the stream priority values and descriptor addresses, which are updated every scheduler cycle. As scheduling decisions are made on a frame-by-frame basis, data caching has the effect of reducing the "Total Scheduling time" by $\approx 2182\mu$s and $\approx 2130\mu$s, respectively, for the software FP and fixed-point implementations. Scheduler decision latency is $\approx 67\mu$s (i.e., the difference between "Average frame Sched time" and "Average frame time without the scheduler") for the fixed-point version.

1. Data caching has to be explicitly enabled on our i960RD NI because the VxWorks disk driver currently supports disk accesses only with data cache disabled (the disk driver disables the data cache automatically on reboot). Therefore, for the measurements in Table 1, we first read the MPEG file from the NI-attached disk and populate the NI-resident circular buffer. After this, we enable the data cache, since further accesses to the locally attached disk are not required.

3.2.4 Runtime NI Extension Is Important

Data caching is one reason for using multiple NIs with each cluster node. By dedicating an NI to a specific task (i.e., scheduling versus disk access) and thereby separating the NIs that produce media streams resident on attached disks from NIs that schedule such streams, scarce NI resources can be specialized to perform these tasks efficiently. For the i960RD NIs, this means that the scheduler thread can benefit from data caching without being limited by the disk driver that disables the data cache [63], [62], [21], [38], [64]. More generally, this fact indicates that the software architecture of NIs must permit the runtime *extension* of NI functionality so that NIs can be dynamically specialized for the diverse tasks they must perform on behalf of applications running on the host nodes they are connecting [21]. A simple extension interface for an NI is described in [45]. Our ongoing work is generalizing this interface to dynamically configure the NI's software and its attached FPGA or other specialized stream processing hardware.

3.2.5 Hardware Queuing Is of Limited Utility

The i960RD cards provide a number of hardware resources for device operation. These include outbound and inbound circular queues and index registers. The "Hardware Queues" on the i960RD card are a set of 1,004 32-bit memory-mapped registers in local card address space. Accesses to the memory-mapped registers do not generate any external bus cycles (off-processor core). To investigate whether indexing into a circular buffer of frame descriptors may be done faster if their addresses reside in memory-mapped register space, we implemented a circular buffer where each 32-bit register holds the descriptor (with address and other attributes) of an MPEG frame. Measurements similar to Table 1 were completed. The results described in [28] indicate no additional performance advantages derived from hardware-based descriptor queues and, more generally, they again demonstrate the relative paucity of NI-based hardware resources suitable for wide ranges of application-specific computations. We derive from this paucity the need for dynamic specialization and tuning of application-specific computations when they are mapped to NIs. We have had similar experiences with more modern NIs, like Alteon's gigabit Ethernet boards [55], and even with "richer" boards, such as Intel's IXP 1200 router boards [52], [18], [68].

3.2.6 Future NIs Can Schedule Media Frames for Gigabit Links

We have demonstrated that card-to-card PCI transfers may be completed without host involvement, thereby making the use of multiple NIs on a single host advantageous and

TABLE 2
Scheduler Latency for Different Processor Configurations

Processor	Scheduler Latency(μs)
i960RD 66MHz, 4K I-cache 2K D-cache, 44MHz memory bus (with OS)	67
StrongArm SA-110 85.7MHz, 16k I-&D-cache, 28.6MHz memory bus (no OS)	17.48
StrongArm SA-110 287MHz, 16k I-&D-cache, 95.7MHz memory bus (no OS)	5.21

also leaving the host CPU free to do other tasks [63], [62], [64]. We have established that it is feasible to offload scheduling functionality from the host to NIs while still meeting the frame-time requirements of MPEG frames. In general, for a scheduling discipline to schedule packets at wire-speeds and to achieve full-link utilization, decisions must be completed within Ethernet frame times. Results from this section show that even on a 66 MHz i960RD processor, the scheduler can pick winner-entities in $\approx 65\mu$s, which is within an Ethernet frame time of 120μs at 100Mbps for a 1,500 byte frame.

For full (max MTU) Ethernet frames, the i960-based CoProcessors can handle per-frame scheduling for up to 100Mbps Ethernet links. Furthermore, larger scheduling units like MPEG-I frames (each comprised of multiple Ethernet frames) may be schedulable even for gigabit Ethernet links.

A number of vendors are including faster and richer CoProcessors on NI boards. Alteon [55] AceNic includes two MIPS cores, and the IXP1200 [52] from Intel has a StrongArm core and six hardware RISC microengines clocked at 200MHz. With substantially faster processors, it is likely that media schedulers running on these boards can meet the packet-time requirements of Ethernet frames at gigabit link speeds, as indicated by initial results described in [68] and Table 2.

Table 2 shows the results from running the packet scheduler (similar conditions as in Table 1, i.e., with data-cache support and integer version of the packet scheduling algorithm) on a StrongArm simulator (evaluation version 1.1 from ARM) for two different configurations. Both configurations have compiled code (ARM gcc compiler flag -O1) being run directly by the simulator without any operating system (no external interrupts but with memory management). These configurations are similar to code running directly on the IXP1200 RISC microengines (without operating system support). Results from the original i960 configuration are shown in Table 2 for comparison (with VxWorks support). A StrongArm SA-110 configuration with processor and memory speeds similar to the i960 configuration yields a scheduler latency of $\approx 17.48\mu$s, which is substantially lower than the i960 scheduler latency. The third configuration is similar to the IXP1200 RISC microengines (clocked at 200MHz without any OS support) at 287 MHz and with a memory bus speed of 95.7 MHz. This shows a scheduler latency of $\approx 5.21\mu$s, thereby capable of scheduling at least 1,500-byte frames at gigabit link rates. We expect to be able to lower this with careful optimization of code and compiler-assisted data placement in cache (also see [68]). We conclude from these measurements that software versions of the DWCS packet scheduler hold great promise in supporting gigabit link scheduling using modern processor chips.

3.2.7 Multiple Gigabit Links or 10Gbps Links Require Custom Packet Scheduling Hardware

Current servers already support two to three 1-Gbps interfaces to match the backplane bandwidth of the internal PCI backplane interconnect (4.2Gbps). With the arrival of 10Gbps Infiniband and 10Gbps Ethernet hardware ([3]), the software solutions for packet scheduling explored in this paper are unlikely to scale to large numbers of packet streams. A follow-up research effort has completed construction of a custom hardware scheduler for 10Gbps links. The ShareStreams hardware prototype consists of a Xilinx Virtex I/Virtex II CoProcessor for real-time streaming under host processor systems software control [30]. We refer the reader to [30] for details regarding the systems and hardware architecture and results attained for the Virtex II FPGA chip. Insights depicted in Table 3 demonstrate that custom hardware support is needed for scalability. With such custom hardware, even for 1,024 streams, the scheduler latency can approach 1,500-byte packet-times for 10Gbps links (1.2μs).

An issue for custom scheduling CoProcessors not yet addressed in this paper is that considerable input bandwidth is required to the packet scheduling hardware unit as arrival-times for packets must be provided to the scheduler hardware unit (32-bit values) every 1.2μs. PCI-66MHz (4.2Gbps peak) buses and PCIX-133MHz (8.5Gbps peak) buses do not have the backplane bandwidth to support 10Gbps links. We expect upcoming Infiniband intracomputer (system internal) interconnects to provide the internal bandwidth to support 10Gbps NIs [7]. For current Gbps NIs however, enough internal bandwidth exists with PCI and PCI-X to support multi-Gbps cross-traffic and transmission of packet-arrival times to a specialized scheduler hardware unit.

TABLE 3
Scheduler Latency with Virtex II FPGA Hardware

Stream Count	Scheduler Latency (μs)	Comments
4	0.0397	meets 10Gbps packet-times
32	0.132	meets 10Gbps packet-times
1024	1.65	nearly meets 10Gbps packet-times

TABLE 4
Critical Path Benchmarks

Expt	Frame Transfer Path (1000 byte frame)	Frame Transfer Time (msec)
I	Disk-Host CPU-I/O Bus-Network (no load w/ cache)	1(ufs)/8(VxWorks)
II	NI Disk-NI CPU-Network (no cache)	5.4
III	Disk-I/O Bus-NI CPU-Network (no cache)	5.415 (4.2disk+1.2net+0.015pci)

3.3 Media Scheduling and Streaming on NIs: Opportunities and Advantages

CoProcessors can affect media scheduling in ways other than filtering frames in real-time and the consequent removal of load from cluster nodes' I/O infrastructures, CPUs, and memories. Next, we evaluate the ability of NIs to perform more complex tasks, such as the retrieval of media content from disks directly attached to them. We also demonstrate that media scheduling on a CoProcessor is not subject to perturbation caused by system overloads. An intuitive reason for this fact is the relatively simpler nature of NIs compared to host hardware and software. This fact, coupled with the smaller, only slowly changing set of tasks a typical NI performs, makes it easier to add functionality to the NI without perturbing its other tasks or, at minimum, it makes it easier to diagnose the degree of perturbation such tasks will experience. This is not true for host CPUs, even when they are multiprocessors like the quad Pentium Pro machines used in this research, when processors are dedicated to run stream schedulers and when using all means possible to separate host-resident stream schedulers from other host tasks. Experimental results validating this claim appear later in this section.

3.3.1 *There Are Multiple Alternatives in Media Streaming from NI-Attached Disks*

One promise of server-attached I/O CoProcessors is that simple tasks can be performed entirely independently of host CPUs, thus freeing them for other duties and/or creating a more scalable server system comprised of the host and a number of relatively low-cost I/O CoProcessors. For high performance applications, researchers have derived benefits from the concurrency and asynchrony of execution of host versus NI tasks and from the low-latency access most NIs have to the actual network transceiver [45], [48], [37]. For media scheduling, we next evaluate the ability of CoProcessors to independently stream media frames to clients once the host CPU has identified the appropriate media files and their storage or remote sources. The experiments shown below evaluate 1) whether a single NI can be both a data source and perform media scheduling, 2) how two NIs, one acting as the data source, the other performing scheduling, compare in performance to the first configuration and, finally, 3) whether the high levels of performance offered by host-resident file systems can be matched by NI-attached disks and their file system support. Option 1) is feasible for our NIs because each i960RD card has two SCSI ports and two 100 Mbps Ethernet ports, as shown in Fig. 1. Disks may be attached to the card's SCSI ports and media may be streamed directly to the network using the card's 100 Mbps Ethernet port. We also investigate Option 2) because, for other NIs, we can assume the presence of peer-to-peer PCI support and an ability to change firmware to accommodate media scheduling, but disk accesses may have to be performed either via network-attached disk devices (i.e., media data is streamed to the NI via a network link) or from an intelligent disk controller [1]. Our experience is that, for buses like PCI with multi-transaction timers and interdevice addressing capabilities, peer-to-peer transfers can be performed in an efficient manner. We note that both for Options 1) and 2), host CPUs like UltraSparcs and Pentium IIs are insulated from the I/O bus transfers involved through the UltraSparc Data buffer (UDB) and the PCIset, respectively.

3.3.2 *NIs Can Stream Data from Their Attached Disks with Performance Comparable to that Achieved by Hosts*

This statement is validated experimentally in the remainder of this section. Specifically, consider a host CPU-based scheduler that uses host filesystem buffers to store media frames, consumes I/O and system bus bandwidth, host memory, and kernel/user space buffers for dispatching frames to the network. An MPEG file resident on disk must be transferred to the host's filesystem buffers by the disk controller via the I/O bus. This is shown as Path A in Fig. 2. In comparison, a network interface card with an attached disk acting as a media source can use peer-to-peer PCI transfers to move data from disk, to scheduler input queues, to the network, thereby eliminating the use of host-based resources, including the host system bus. This is shown as Path B in Fig. 2. Path C streams frames from a disk directly attached to the NI through to the network, eliminating PCI I/O bus bandwidth consumption, host-bus bandwidth, and host-memory resources in Fig. 2.

Next, we present results from critical path benchmarks recorded for three different configurations of frame transfers. All benchmarks measure the latency of a 1,000 byte frame transfer from disk to remote client (over an Ethernet network) averaged over 1,000 transfers. The same physical disk device is used in all experiments. The measurements in Table 4 record the latency of a single frame transfer.

NI-attached disks approximate the performance of fast host I/O systems. Consider Experiment I in Table 4, represented by Path A in Fig. 2. An MPEG file on an internal system disk attached to a disk controller on the PCI bus is streamed to a remote client. The system disk is attached to an SCSI controller card on the PCI bus. The results, shown in Table 4, Experiment I, show a total frame transfer time via the network of 8 ms (including disk access

latency) when using the VxWorks filesystem on the Solaris host. This compares unfavorably with the results in Table 4, Experiment II, where a total of 5.4 ms is required to perform the same action for a file already resident on the NI's attached disk (Path C in Fig. 2). It does indicate, however, that scalability in the number of media streams serviced by a single host can be improved by the addition of low-cost I/O CoProcessors directly attached to the network. These results for Experiment I were obtained on Solaris 2.5.1 with an Intel 82557-based NI, which has the same Ethernet transceiver chipset as the i960RD CoProcessor. The system disk attached to the disk controller was used to serve frames using the Intel NI. The VxWorks filesystem is a DOS-based filesystem and this was mounted on the Solaris host in order to mitigate the effects of variations in file system performance and disk layout. Thus, the latency component common to Experiments I, II, and III is the disk access time, which is $\approx 4.2ms$ for a single frame (see the value "4.2 disk" in Table 4).

The advantages of NI-attached disks are reduced substantially when using the faster Solaris UFS filesystem on the host. In this configuration, Experiment I experiences a disk frame latency of only 1 ms due to the larger logical block size (8K) used by UFS and its disk block caching and prefetching enabled by the host's large main memory. VxWorks does not support the UFS filesystem so that we were unable to mount it on the NIs for Experiments II and III [13], [38], but we hypothesize that its use would improve the performance of Experiments II and III substantially if the NIs have large disk buffer caches. While this is one strong recommendation we derive from this research for NI-based media access, we also note that the host-based performance advantages gained from buffering do not extend to live media (i.e., media captured and distributed in real-time). Also, the NI CoProcessor can be allowed access to the filesystem disk block buffer cache *on the host* so that frames can be accessed directly by the CoProcessor and streamed to the network using I/O bus DMAs. The NI CoProcessor can use the host filesystem buffer cache *directly* and leverage the more efficient host filesystem for bulk data transfers. This can help alleviate the shortcomings of the NI attached filesystem in a substantial way.

It is advantageous to use multiple NIs, each with specialized tasks. A property of CoProcessors is that the richer their functionality, the larger the latencies of their message transfers and, possibly, the smaller the total message throughput they support (see Section 3.2) [54], [37]. Our approach to this problem is to specialize NIs such that each NI only performs a limited number of tasks. In the case of media streams, this means that one NI has an attached disk; the other acts as a media scheduler and network gateway. Only small additional overheads are experienced by this NI configuration (also see Path B in Fig. 2). Specifically, for a single media frame, the required peer-to-peer PCI transfers add only about 15μs to the total time of accessing a frame, scheduling it, and dispatching it to the network (we used DMA writes from card-to-card to achieve this). We achieved transfers of 66.27 MB/s along with PIO read and write latencies of 3.6μs and 3.1μs on a 32-bit 33MHz PCI bus.

The experimental results described thus far have demonstrated the viability of NI-based disk attachment and media streaming, which effectively removes the host CPU from the execution of such relatively straightforward server actions.

NI-based streaming has inherent performance advantages. To generalize our cost arguments concerning NI-based data streaming and scheduling, consider that the action of streaming frames involves 1) selecting a stream from a set of streams (sched), 2) accessing a frame from the selected stream (access), and 3) transferring the frame to the client (transfer). Stated more formally, let t_{sched} denote the time to select a stream and $t_{delivery}$ denote the time taken to deliver a single frame from the selected stream to the output link. Let t_{access} be the time to access the frame (from storage) and $t_{transfer}$ denote the transfer latency to the output link. Then:

$$t_{delivery} = t_{access} + t_{transfer}$$

and

$$t_{stream} = t_{sched} + t_{delivery},$$

where t_{stream} is the total time for streaming a single frame.

In the case of *host-based* streaming, $t_{delivery}$ involves the relatively complex Path A shown in Fig. 2, whereas, for a CoProcessor, $t_{delivery}$ involves the less complex Paths B or C in Fig. 2, the latter not involving any host/IO bus domain traversals. In conditions of low load and extrapolating from the measurements depicted in Table 4 (assuming the same storage source and file system capabilities for both CoProcessor and host), t_{stream} for host-based streaming is ≈ 8.050 ms (host scheduler overhead is $\approx 50\mu$s) and $\approx 5.465ms$ for CoProcessor-based streaming for a single 1,000 byte frame. The advantages of CoProcessor-based streaming are exacerbated when there are host-CPU loads, as shown in Section 3.4, since t_{stream} for a host can experience significant increases even for transient loads. Further, note that t_{sched} is necessarily experienced by every frame that requires scheduling, as stream selection is done on a packet-basis and, therefore, cannot be pipelined. As a result, if the term $t_{stream}^{HostLoad}$ represents t_{stream} with host load, then, for a host CPU, $t_{stream}^{HostLoad} > t_{stream}$. On the other hand, for an NI CPU, host load will not affect t_{stream}, as evident in Path C in Fig. 2.

3.4 Perturbation of NI versus Host-Based Scheduling under Load

3.4.1 NI-Based Scheduling Has Reduced Delay-Jitter

The previous section has already argued that a packet scheduler running on the host (i.e., see Path A in Fig. 2) uses the host's memory, bus, and I/O bus resources. In comparison, NI-based scheduling (see Paths B or C in Fig. 2) may completely avoid the consumption of I/O bus bandwidth (see Path C in Fig. 2) and never uses host memory and bus bandwidth. It will, therefore, be unaffected by host CPU load. This is in stark contrast to host-based scheduling, which is easily affected by the host OS's need to run higher-level application services, where even a minimal OS installation runs system daemons and where media servers maintain meta-information in additional servers or databases. This section demonstrates experimentally the effects of CPU

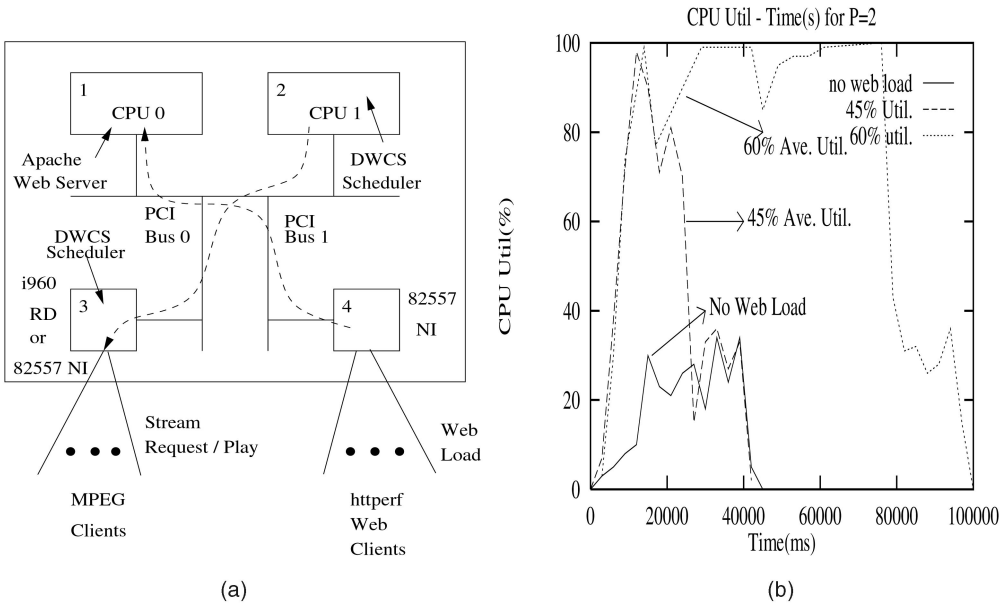


Fig. 4. (a) Server loading architecture—Web and media traffic. (b) CPU utilization variation with server load.

contention on host-based scheduling, where degradation is measured as a decrease in output bandwidth available for a stream and as an increase in its frame queuing delay. Specifically, when the DWCS scheduler receives CPU service at lower rates because of increased service load, that will lead to back-logged frames in scheduler input queues, which in turn causes missed deadlines and loss-tolerance violations. The resulting packet-dropping leads to lower scheduling quality. This is particularly important for jitter-sensitive, live-media traffic, where undue variation of the rate at which the frame/packet scheduler receives CPU services may further increase delay-jitter. Additional delay-jitter in frame output is caused by waits on congested resources, such as the multiple bus/network scheduling domains (system bus to I/O bus and then to the network) and memory hierarchies that must be traversed by host-scheduled streams.

3.4.2 Experimental Demonstration of Perturbation of Host-Based Scheduling in Loaded Conditions

Experimental Setup. The experimental setup consists of a Quad Pentium Pro server (4 X 200MHz CPUs) running Solaris 2.7 x86 with 128 MB of memory. This machine has two separate PCI bus segments and we place NIs on each of the PCI bus segments, as shown in Fig. 4.

For host-based scheduling load experiments, Intel 82557 100Mbps transceiver-based NIs are placed in separate slots on each of the two bus segments (components 3 and 4 in Fig. 4). For NI-based scheduling load experiments, one of the Intel 82557 NIs is replaced with a i960RD NI (component 3 in Fig. 4). The machine runs the Apache Web server version 1.3.12 (with a maximum of 10 server processes and starting process pool with five server processes) [6]. The Web server is loaded using "httperf" (version 0.6) [36] run on remote Linux-based clients. Flexible specification of load from remote clients is allowed by "httperf," where Web pages may be requested at a certain rate by a number of connections, with a user-specified ceiling on the total number of calls. The experimental infrastructure is shown in Fig. 4. Placing two NIs on different PCI bus segments separates Web load and stream traffic. One of the NIs (with IP address bound to the Intel 82557-based NI) is used to load the Web server using "httperf" client traffic, while the other NI (with a different IP address bound to the NI, 82557-based or i960RD NI) is used to request and source stream traffic.

Moderate system loads have substantial effects on host-based scheduling. The first set of experiments involves the host-based scheduler version of the DWCS algorithm. For these experiments, two of the CPUs are brought offline, for a total of two online CPUs. The Apache Web server in the configuration described above is brought up and bound to the IP address of one of the NIs. The DWCS scheduler is initiated and bound to one of the host's processors, using the "pbind" Solaris facility [13]. Client requests are accepted on a separate IP address bound to a different NI. This allows "httperf" Web clients to connect and load the Apache Web server using a specific IP address bound to a specific NI. Similarly, MPEG clients may connect to a different IP address, again bound to a different NI, for stream delivery. This experiment involves Components 1, 2, 3, and 4, as shown in Fig. 4, with Component 3 as an Intel 82557 NI. Two MPEG clients shown as Streams s1 and s2 (in Fig. 5) connect to the system.

These server-load experiments demonstrate that even moderate server loads can result in substantial variations in frame bandwidth and delay. Consider, for instance, the CPU utilization depicted in Fig. 4b (measured using Solaris' Perfmeter facility) [13], which is the load experienced when the host-based scheduler is run with and without any load imposed by the remote Web clients. With no Web load and peak utilization around 35 percent, with an average utilization of 15 percent, the corresponding variations in bandwidth and mean queuing delay experienced for two streams s1 and s2 appear in Figs. 5a and 5b (see the entries labeled "no Web load"). In comparison, even a moderate

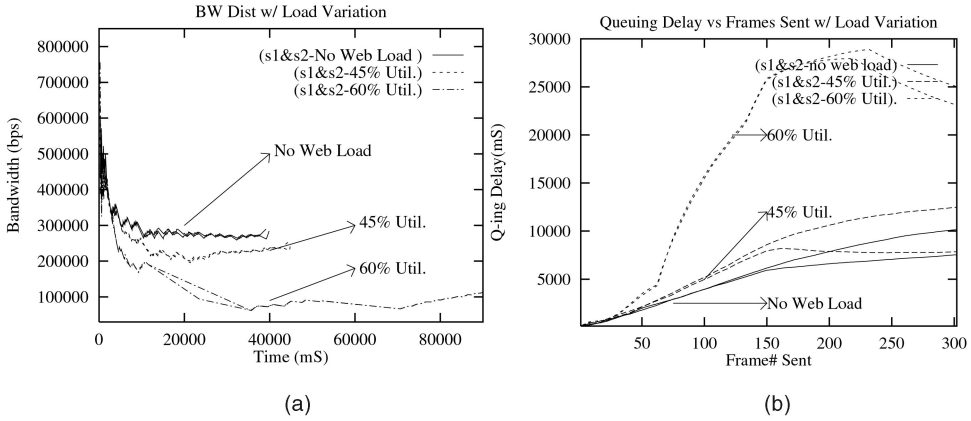


Fig. 5. (a) Bandwidth variation with load. (b) Queuing delay variation with load.

additional host load, when applying load from Web clients at the 45 percent utilization level, leads to noticeable differences in the observed variations in bandwidth and queuing delay. Specifically, at the 45 percent load level, a decrease in bandwidth to 200,000 bps is seen at the 15s-20s time mark and the bandwidth settles at only 230,000 bps (see Fig. 5). The queuing delay graph in Fig. 5b also shows the effects of loading, with frames suffering additional queuing delay of around 2s in the presence of load. A substantial Web load applied at the 60 percent level results in what may be considered undue levels of variation. At the 60 percent average load level, severe degradation is seen. For instance, Fig. 5 shows a decrease in bandwidth to around 100,000 bps when the CPU utilization is in the excess of 80 percent (see Fig. 4b) during the period from 40s-80s. The bandwidth settles to less than 125,000 bps—half of the bandwidth seen in the absence of Web server load (see Fig. 5). Frames can experience excessive queuing delay in the presence of load (60 percent average utilization), up to three times (30,000 ms) than seen in the absence of load (10,000 ms).

NI-based scheduling is not affected by host loading. The next set of experiments involves the NI-based scheduler. For purposes of this experiment, one CPU is brought offline (for a total of one online CPU), with one 82557-based Intel NI used for Web server loading and a i960RD-based NI used for MPEG streaming (DWCS runs on the NI). The i960RD NI is placed on a separate bus segment and MPEG frames are streamed to clients by the NI-based scheduler. Loading of the Web server is done using the other NI placed on a separate bus segment.

This experiment involves components 1, 3, and 4, with component 3 as an i960RD NI that directly streams media content to clients. Initially, streams are played to MPEG clients in the absence of any Web server load, with bandwidth variations and queuing delay measured on the NI-CPU. Next, the system is loaded using the load profile shown in Fig. 4b (for 60 percent average utilization).

The measurements depicted in Figs. 6a and 6b demonstrate that the NI-based scheduler is immune to Web server loading. In addition, no noticeable variations in bandwidth and queuing delay are experienced (see Figs. 6a and 6b) for streams s1 and s2, for loaded and unloaded servers). A settling bandwidth of around 260,000 bps is observed for stream s1, which is similar to the settling bandwidth achieved by the host-CPU based scheduler in the absence of load (250,000 bps in Fig. 5). These results also indicate that the i960RD NIs have sufficient "horse-power" to stream scheduled media frames to clients at real-time rates.

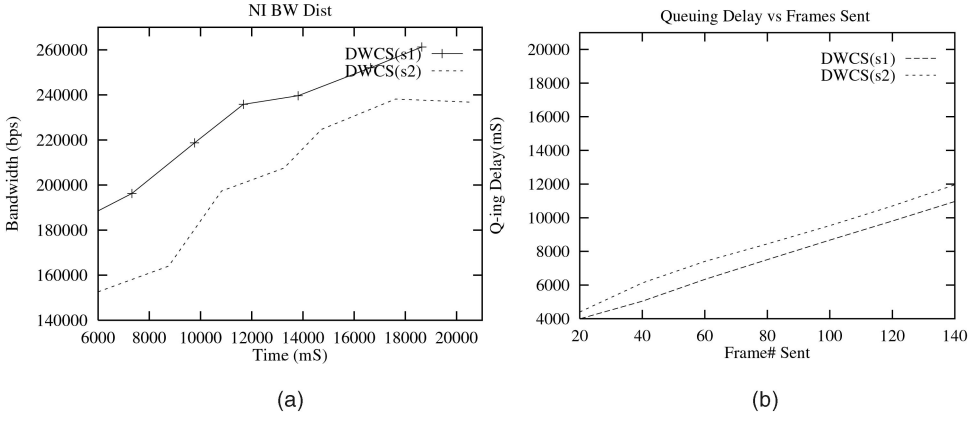


Fig. 6. (a) NI Bandwidth distribution snapshot: unaffected by system load. (b) NI Queuing delay snapshot: unaffected by system load.

4 Related Work

4.1 Communication CoProcessors

A number of NI-based research projects have focused on providing low-latency message passing over cluster interconnects like ATM, Myrinet, FDDI, and HIPPI [15], [40], [58], [57] by using intelligent NIs equipped with programmable CoProcessors [10], [21], [38], [45], [49]. In part, such work is motivated by the presence of programmable CoProcessors in NIs. More importantly, past research has demonstrated improved performance derived from NI reprogramming for control operations that touch multiple machines like collective communications [53], transactions [48], barriers [37], [45], and simpler cluster-wide synchronization constructs [48]. NIs have also been programmed to provide new services, like performance monitoring [34] or like the implementation of portions of protocol stacks on NIs [12].

Newer hardware developments present an interesting perspective on such research. Namely, with gigabit Ethernet [55] as with the higher throughput smart port cards being developed to power the Internet's switching [32] and routing infrastructure, programmable CoProcessors (sometimes even enhanced by configurable hardware like FPGAs [30]) are becoming increasingly common. This trend coupled with the decreasing costs of processor chips suggests that future NI or I/O processor hardware will have substantial CPU cycles and memory with which additional services may be implemented. One such product is Intel's IXP 1200 router board [52], which we have begun to use as an intelligent CoProcessor for a cluster machine [68], [18].

Our work leverages prior efforts and current hardware trends by assuming that CoProcessors cannot be enhanced to provide all desired additional services at all times. Instead, their relative resource paucity indicates the importance of offering, for CoProcessors, runtime extension interfaces via which the appropriate functionality can be placed onto NIs at the right times. One such interface is described in our prior research with FORE SBA-200 (i960CA) cards [45], another in the SPINE project [17], the latter also addressing safety issues for such extensions.

The I2O industry consortium defined a specification for development of I/O hardware and software to allow portable device driver development by defining a message-passing protocol between the host and peer I/O devices [21], [38], [64]. The focus was on relieving the host from tasks that may be offloaded to a programmable NI. Industry efforts included I2O cards for RAID storage subsystems and off-loading TCP/IP protocol processing to the NI from the host [21], [38], [64]. While this paper uses sample I2O cards for the experimental results being presented, we do not rely on the I2O communication standard, but instead, simply use the card's CoProcessor resources. In contrast, the results presented here could benefit from the presence of improved CoProcessor-Host connectivity, as promised by the Infiniband standard [7], which offers intracomputer (within a system) and intercomputer (between systems) interconnects at 2.5 Gbps, 10 Gbps, and 30 Gbps. Improved intracomputer bandwidth using an Infiniband crossbar would help for several reasons. First, additional bandwidth from the memory subsystem to the NI would enable us to store packets in multiple places, including both the host memory subsystem or NI memory, with sufficient bandwidth available to stream packets directly from memory to output link(s). Second, storage controller channel adaptors could provide stream blocks directly to the NI using the crossbar interconnect, as the storage controller and the NI would be peers on the same Infiniband crossbar ports. The high bandwidth of an Infiniband intercomputer network would promote scalability by allowing many streams to be serviced at output link wire-speeds. In general, concerning intelligent CoProcessors in general, high performance intraprocessor interconnects like Infiniband help bridge the "performance wall" between CPU, I/O devices, and the host processor/memory subsystem. As a result, intelligent NI-based CoProcessors would be better able to use their dedicated CPU resources to provide predictable, dedicated delivery of media streams. Similarly, if the CoProcessor is simply a scheduling engine with media stream access and delivery performed by the host, for packet scheduling to be performed at wire speeds, the host and CoProcessor must exchange packet-arrival times of streams in every decision cycle. Even this exchange requires high aggregate bandwidth from the memory subsystem, involving 32 bits in the best case and $1,023 \times 32$ bits (if 1,023 streams drop their packets) in the worst-case for 1,024 streams, every 1.2μs (for 10 Gbps links). With RAMBUS [43] memory systems, these bandwidth requirements are only met for the best case (4.2GBytes/sec) in current systems. Current PCI-66MHz and PCIX-133MHz cannot provide the required bandwidth, whereas Infiniband's internal crossbars should offer the capabilities suitable for scaling to future increases in network link bandwidth.

Analogous to our work with extensible NIs, there has also been research on extensible I/O CoProcessors, as with efforts like Active Disks [1] or extensible network-attached stores [31].

4.2 Media Stream Scheduling

Recent research has put substantial effort into the development of efficient scheduling algorithms for media applications. Given the presence of some underlying bandwidth reservation scheme, the DWCS algorithm has the ability to share bandwidth among competing clients in strict proportion to their deadlines and loss tolerances. In comparison, fair share scheduling algorithms [66], [41] (some of which are now implemented in hardware [44]) attempt to allocate $1/N$ of the available bandwidth among N streams or flows. Any idle time, due to one or more flows using less than their allocated bandwidth, is divided equally among the remaining flows. This concept generalizes to weighted fairness in which bandwidth must be allocated in proportion to the *weights* associated with individual flows, but packet deadlines are not taken into account. We, therefore, consider DWCS preferable for the media streams addressed by our work.

There has been significant research on the construction of scalable media servers and services, including recent work on reservation-based CPU schedulers for media applications [24]. These results demonstrate the importance of explicit scheduling to meet the demands of media applications. If DWCS performed its scheduling actions using a reservation-based CPU scheduler, it would be able to closely couple its CPU-bound packet generation and scheduling actions with the packet transmission actions required for packet streams. Similarly, DWCS could also take advantage of the stripe-based disk and machine scheduling methods advocated by

some video servers [11] by using stripes as coarse-grain "reservations" for which individual packets are scheduled to stay within the bounds defined by these reservations.

5 Conclusions and Future Work

The vision pursued by our research is one in which underlying hardware/software platforms, like the communication CoProcessors used here, are dynamically extended to better meet the needs of applications. This paper's demonstration of our vision realizes a media scheduler on an embedded NI CoProcessor using commodity hardware and software.

Insights derived from our experimental research include the following:

- *Efficient execution on standard NIs.* Even commodity NI hardware has resources sufficient for handling both the regular tasks it must perform (e.g., message receipt and sending) and certain additional tasks required by individual applications. In fact, experimental results attained with the DWCS packet scheduler for media applications demonstrate that even older, relatively slow i960-based NIs can perform a variety of such tasks, at real-time rates and at a granularity of scheduling suitable for MPEG media streams.
- *Improved scalability and predictability for NI-based streaming services.* By running media scheduling services on NIs rather than host CPUs, the host's CPU, memory, and I/O infrastructure are offloaded and, in addition, substantial improvements are attained in the predictability of media streaming, measured as improvements in the delay-jitter of media streams.
- *Performance improvements due to traffic elimination.* An advantage derived from placing media scheduling onto NIs is the elimination of traffic from the host node. This fact is strengthened by the "filtering" property of media scheduling in which losses are allowed. This property is shared by many other applications being investigated in our current research, including the selection of subsets from sensor data or from large scientific or engineering data, and data downsampling or compression. The latter typically require more processing power than is currently available on commercial NIs.

To summarize, the approach to scalability for media streaming advocated by this paper has three components. The first is scalable scheduler algorithm design [63], [62] and the efficient realizations of such algorithms. The second is the appropriate distribution of media scheduling across available hardware/software resources (in this case, i960RD cards, PCI bus segments, and disks) in order to separate the resources used for media scheduling from those used by general host applications. This approach also ensures that host-based programs and loading conditions do not interfere with NI-based media streaming and scheduling.

5.1 Future Work

One insight from this paper is that CoProcessors cannot be assumed capable of scheduling packets at the per-frame rates required by future communication links (e.g., 10Gbps Ethernet (10GEA standard) or Infiniband). To address this issue, we have built QoS packet scheduling hardware architectures that can meet the wire-speeds of 10Gbps links. The FPGA hardware CoProcessor can meet the wire-speeds of 10Gbps links under systems software control of a peer or Host-NI CoProcessor. We are currently investigating scaling of this architecture to a large number of streams by allowing stream state to be multiplexed on the same FPGA hardware substrate [60], [26], [27], [30]. In addition, we are studying how to "divide" scheduling functionality or how to simplify it so that per frame scheduling may be performed with the RISC-based microengines resident on typical next generation CoProcessors, such as Intel's IXP1200 [68]. Finally, beyond packet scheduling, we are investigating suitable extension architectures for NIs, so that application-specific functionality is easily mapped to them, whenever applications can benefit from such NI-based support. One set of extensions already under development by our group concerns the reliability and scalability of transaction services implemented on cluster machines [19], [20].

Acknowledgments

This work was supported in part by the US Department of Energy under its NGI program, by the US National Science Foundation, and by hardware/software donations from Intel Corporation and WindRiver Systems. The authors would like to thank the reviewers for their many invaluable suggestions. Their thanks also go to other contributors to this research, including Professor Sudhakar Yalamanchili, Professor Ken Mackenzie, and MS students S. Manni and S. Roy.

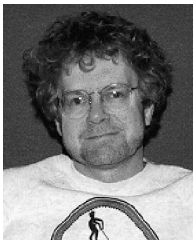
References

[1] A. Acharya, M. Uysal, and J. Saltz, "Active Disks," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 1998.
[2] Akamai and FreeFlow Content Management, http://www.akamai.com, 2003.
[3] 10 Gigabit Ethernet Alliance, http://www.10gea.org, 2003.
[4] K. Almeroth and M. Ammar, "A Scalable, Interactive Video-on-Demand Service Using Multicast Communication," *Proc. Int'l Conf. Computer Comm. Networks,* Sept. 1994.
[5] T.E. Anderson, D.E. Culler, D.A. Patterson, and the NOW Team "A Case for Networks of Workstations: NOW," *IEEE Micro,* Feb. 1995.
[6] Apache http Server Project Apache Software Foundation, http://www.apache.org/httpd.html, 2003.
[7] Infiniband Trade Association, http://www.infinibandta.org, 2003.
[8] B.N. Bershad, S. Savage, P. Pardyak, E.G. Sirer, M. Fiuczynski, and B. Eggers Chambers, "Extensibility, Safety, and Performance in the SPIN Operating System," *Proc. 15th ACM Symp. Operating Systems Principles,* pp. 267-284, Dec. 1995.
[9] B. Blake, "A Fast, Efficient Scheduling Framework for Parallel Computing Systems," PhD thesis, Dept. of Computer and Information Science, The Ohio State Univ., Dec. 1989.
[10] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.-K. Su, "Myrinet—A Gigabit-per-Second Local-Area Network," *IEEE MICRO,* Feb. 1995.
[11] W.J. Bolosky, R.P. Fitzgerald, and J.R. Douceur, "Distributed Schedule Management in the Tiger Video Fileserver," *Proc. 16th ACM Symp. Operating System Principles,* vol. 31, pp. 212-223, Dec. 1997.

[12] C. Keppitiyagama, et al. "Asynchronous MPI Messaging on Myrinet," *Proc. Int'l Parallel and Distributed Processing Symp.,* Apr. 2001.
[13] Solaris On-Line Documentation, http://www.docs.sun.com, 2003.
[14] D.R. Engler, M.F. Kaashoek, and J. O'Toole Jr., "Exokernel: An Operating System Architecture for Application-Level Resource Management," *Proc. 15th Symp. Operating System Principles,* Dec. 1995.
[15] E.W. Felten, R.D. Alpert, A. Bilas, M.A. Blumrich, D.W. Clark, S. Damianakis, C. Dubnicki, L. Iftode, and K. Li, "Early Experience with Message-Passing on the SHRIMP Multicomputer," *Proc. 23rd Int'l Symp. Computer Architecture,* May 1996.
[16] D. Ferrari, A. Banerjea, and H. Zhang, "Network Support for Multimedia—A Discussion of the Tenet Approach," TR-92-072, Dept. of Computer Science, Univ. of California Berkeley, 1992.
[17] M.E. Fiuczynski, B.N. Bershad, R.P. Martin, and D.E. Culler, "SPINE—An Operating System for Intelligent Network Adapters," TR-98-08-01, Aug. 1998.
[18] A. Gavrilovska, K. Schwan, A. McDonald, and K. Mackenzie, "Stream Handlers: Application-Specific Message Services on Attached Network Processors," *Proc. 10th IEEE Conf. High-Performance Interconnects,* Aug. 2002.
[19] A. Gavrilovska, K. Schwan, and V. Oleson, "Adaptable Mirroring on Clusters," *Proc. 10th Int'l Conf. High Performance Distributed Computing,* Aug. 2001.
[20] A. Gavrilovska, K. Schwan, and V. Oleson, "Practical Approach to Zero Downtime in an Operational Information System," *Proc. 22nd IEEE Int'l Symp. Distributed Computing Systems,* 2002.
[21] I_2O Special Interest Group, www.i2osig.org/architecture/techback98.html, 1999.
[22] Intel, *IQ80960Rx Evaluation Platform Board Manual,* Mar. 1997.
[23] C. Isert and K. Schwan, "ACDS: Adapting Computational Data Streams for High Performance," *Proc. Int'l Parallel and Distributed Processing Symp.,* 2000.
[24] M.B. Jones, D. Rosu, and M.-C. Rosu, "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities," *Proc. 16th ACM Symp. Operating System Principles,* vol. 31, pp. 198-211, Dec. 1997.
[25] C. Krasic and J. Walpole, "QoS Scalability for Streamed Media Delivery," Technical Report CSE-99-011, Dept. of Computer Science, Oregon Graduate Inst., 17, 1999.
[26] R. Krishnamurthy, et al. "The Georgia Tech Asan Approach," *Proc. IEEE Int'l Symp. High Performance Computer Architecture,* Jan. 2001.
[27] R. Krishnamurthy, et al. "Architecture and Hardware for Scheduling of Gigabit Packet Streams," *Proc. IEEE Int'l Symp. High Performance Computer Architecture,* Jan. 2001.
[28] R. Krishnamurthy, K. Schwan, R. West, and M. Rosu, "A Network CoProcessor-Based Approach to Scalable Media Streaming in Servers," Technical Report GIT-CC-00-03, Georgia Inst. of Technology, 2000.
[29] R. Krishnamurthy, K. Schwan, R. West, and M. Rosu, "A Network Co-Processor-Based Approach to Scalable Media Streaming in Servers," *Proc. Int'l Conf. Parallel Processing,* Int'l Assoc. for Computers and Comm. (IACC), Aug. 2000.
[30] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West, "Architecture and Hardware for Scheduling Gigabit Packet Streams," *Proc. 10th IEEE Conf. High-Performance Interconnects,* Aug. 2002.
[31] E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating,* 1996.
[32] J. Lockwood, J. Turner, and D. Taylor, "Field Programmable Port Extender (FPX) for Distributed Routing and Queuing," *Proc. ACM Int'l Symp. Field Programmable Gate Arrays,* pp. 137-144, Feb. 2000.
[33] G. Mair, "Telepresence—The Technology and Its Economic and Social Implications," *Proc. IEEE Int'l Symp. Technology and Soc.,* 1997.
[34] M. Martonosi, D. Clark, and M. Mesarina, "The Shrimp Hardware Performance Monitor: Design and Applications," *Proc. 1996 SIGMETRICS Symp. Parallel and Distributed Tools,* 1996.
[35] C.W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reservation for Multimedia Operating Systems," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* May 1994.
[36] D. Mosberger and T. Jin, "Httperf—A Tool for Measuring Web Server Performance," *Proc. 1998 Workshop Internet Server Performance, held in conjunction with Sigmetrics 1998,* June 1998.
[37] J. Nieplocha, et al. "One-Sided Communication on Myrinet-Based SMP Clusters Using the GM Message-Passing Library," *Proc. Workshop Comm. Architectures in Clusters, held in conjunction with Proc. Int'l Parallel and Distributed Processing Symposium,* Apr. 2001.
[38] I_2O Intel Page, http://www.developer.intel.com/iio, 1999.
[39] V. Pai, P. Druschel, and W. Zwaenepoel, "Lo-Lite: A Unified Buffering and Caching System," *Proc. Third Symp. Operating Systems Design and Implementation,* 1999.
[40] S. Pakin, M. Laura, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," *Proc. Supercomputing,* Dec. 1995.
[41] X. Guo, P. Goyal, and H.M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," *Proc. Second Symp. Operating Systems Design and Implementation,* pp. 107-121, 1996.
[42] B. Plale and K. Schwan, "DQUOB: Managing Large Data Flows by Dynamic Embedded Queries," *Proc. IEEE Int'l Symp. High Performance Distributed Computing,* 2000.
[43] RAMBUS, http://www.rambus.com, 2003.
[44] J.L. Rexford, A.G. Greenberg, and F.G. Bonomi, "Hardware-Efficient Fair Queuing Architectures for High-Speed Networks," *Proc. INFOCOMM,* pp. 638-646, Mar. 1996.
[45] M.-C. Rosu, K. Schwan, and R. Fujimoto, "Supporting Parallel Applications on Clusters of Workstations: The Intelligent Network Interface Approach," *Proc. Sixth IEEE Int'l Symp. High Performance Distributed Computing,* Aug. 1997.
[46] D. Rosu, K. Schwan, and S. Yalamanchili, "FARA—A Framework for Adaptive Resource Allocation in Complex Real-Time Systems," *Proc. Fourth IEEE Real-Time Technology and Applications Symp.,* June 1998.
[47] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha, "On Adaptive Resource Allocation for Complex Real-Time Applications," *Proc. 18th IEEE Real-Time Systems Symp.,* Dec. 1997
[48] M.-C. Rosu and K. Schwan, "Sender Coordination in the Distributed Virtual Communication Machine," *Proc. Seventh IEEE Int'l Symp. High Performance Distributed Computing,* 1998.
[49] M.-C. Rosu, K. Schwan, and R. Fujimoto, "Supporting Parallel Applications on Clusters of Workstations: The Virtual Communication Machine-Based Architecture," *Cluster Computing,* vol. 1, pp. 1-17, Jan. 1998.
[50] Y. Saito, B. Bershad, and H. Levy, "Availability and Performance in Porcupine: A Highly Scalable Internet Mail Service," *Proc. 17th ACM Symp. Operating Systems Principles,* Dec. 1999.
[51] K. Schwan and H. Zhou, "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads," *IEEE Trans. Software Eng.,* vol. 18, no. 8, pp. 736-748, Aug. 1992.
[52] Intel IXP 1200 Web Site, http://www.intel.com/design/network/products/npfamily/index.htm, 2003.
[53] C. Stunkel, D. Shea, B. Abali, M. Atkins, C. Bender, D. Grice, P. Hochschild, D. Joseph, B. Nathanson, R. Swetz, R. Stucke, M. Tsao, and P. Varker, "The SP2 Communication Subsystem," technical report, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., http://ibm.tc.cornell.edu, Aug. 1994
[54] R. Swan, S. Fuller, D. Siewiorek, and C. Modular, "Multi-Microprocessor," *Proc. Nat'l Computer Conf.,* vol. 46, pp. 637-644, 1977.
[55] Alteon Web Systems, http://www.alteonWebsystems.com, 2001.
[56] M. Trivedi, B. Hall, G. Kogut, and S. Roche, "Web-Based Teleautonomy and Telepresence," *Proc. SPIE Optical Science and Technology Conf.,* 2000.
[57] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *Proc. 15th ACM Symp. Operating Systems Principles,* Dec. 1995.
[58] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. 19th Int'l Symp. Computer Architecture,* May 1992.
[59] J. Walpole, R. Koster, S. Chen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu, "A Player for Adaptive MPEG Video Streaming over the Internet," *Proc. 26th Applied Imagery Pattern Recognition Workshop,* Oct. 1997.
[60] R. West, R. Krishnamurthy, W. Norton, K. Schwan, S. Yalamanchili, M. Rosu, and S. Chandra, "QUIC: A Quality of Service Network Interface Layer for Communication in NOWS," *Proc. Heterogeneous Computing Workshop, in conjunction with IPPS/SPDP,* Apr. 1999.

[61] R. West and C. Poellabauer, "Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Packet Streams," *Proc. 21st IEEE Int'l Symp. Real-time Systems,* 2000.
[62] R. West and K. Schwan, "Dynamic Window-Constrained Scheduling for Multimedia Applications," *Proc. Sixth Int'l Conf. Multimedia Computing and Systems,* also available as Technical Report: GIT-CC-98-18, Georgia Inst. of Technology, June 1999.
[63] R. West, K. Schwan, and C. Poellabauer, "Scalable Scheduling Support for Loss and Delay Constrained Media Streams," Technical Report GIT-CC-98-29, Georgia Inst. of Technology, 1998.
[64] WindRiver Systems, *VxWorks Reference Manual,* first ed., Feb. 1997.
[65] Xilinx, http://www.xilinx.com, 2003.
[66] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," *Proc. ACM SIGCOMM,* pp. 113-121, Aug. 1991.
[67] D. Zhou and K. Schwan, "Adaptation and Specialization for High Performance Mobile Agents," *Proc. Usenix Conf. Object-Oriented Technologies,* 1999.
[68] X. Zhuang, W. Shi, I. Paul, and K. Schwan, "On the Efficient Implementation of the DWCS Packet Scheduling Algorithm on IXP1200 Network Processors," *Proc. IEEE Int'l Symp. Multimedia Networks and Systems,* 2002.

Raj Krishnamurthy is a PhD degree candidate in the College of Computing at the Georgia Institute of Technology. His graduate degree is in electrical engineering from Georgia Tech. Until December 2002, he was a research scientist with the Center for Experimental Research in Computer Systems at Georgia Tech. His interests are in architecture and VLSI, with a focus on high-speed networks and real-time systems. He is a student member of the IEEE and ACM. He actively consults for startups in embedded systems and technology strategy.



Karsten Schwan received the MSc and PhD degrees from Carnegie-Mellon University in Pittsburgh, Pennsylvania. His PhD research in high performance computing concerned operating and programming systems support for the Cm* multiprocessor. He is a professor in the College of Computing at the Georgia Institute of Technology. He is also the director of the Center for Experimental Research in Computer Systems (CERCS), a GT entity that spans both the College of Computing and the School of Electrical and Computer Engineering. At Ohio State University, he established the PArallel, Real-Time Systems (PARTS) Laboratory conducting research on operating system support for cluster computing and on real-time applications and operating systems for parallel machines and containing both specialized (embedded) and commercially available machines (e.g., Intel hypercube and BBN Butterfly). At Georgia Tech, his work concerns middleware and systems support for grid and pervasive systems, focusing on the interactive nature of modern distributed and parallel applications (e.g., online monitoring and program steering), the real-time nature of pervasive applications (e.g., system support for predictable operation in dynamic environments), and the online configuration and adaptation of application or system components (e.g., configuring communication infrastructures). He is a senior member of the IEEE.

Richard West received the MEng degree in microelectronics and software engineering in 1991 from the University of Newcastle-upon-Tyne, England. He later received both the MS (1998) and PhD (2000) degrees in computer science from the Georgia Institute of Technology. He is currently an assistant professor in the Computer Science Department at Boston University, where his research interests include operating systems, real-time systems, distributed computing, and QoS management.

Marcel-Catalin Rosu received the MS degrees from the Polytechnic University of Bucharest and Cornell University, in 1987 and 1995, respectively, and the PhD degree from the Georgia Institute of Technology in 1999, all in computer science. He is a research staff member in the Internet Infrastructure and Computing Utilities Department at the IBM T.J. Watson Research Center in Yorktown Heights, New York. His research interests include operating system support for cluster computing, processor scheduling, multimedia, Web server performance, and distributed computing.