Coordinated Allocation and Scheduling of Multiple Resources in Real-time Operating Systems

Kartik Gopalan and Kyoung-Don Kang
Computer Science, State University of New York at Binghamton
Binghamton, NY 13902-6000
{kartik,kang}@cs.binghamton.edu

ABSTRACT

Distributed real-time embedded (DRE) systems are key components of critical infrastructure including surveillance, target tracking, electric grid management, traffic control, avionics, and communications systems. They require (1) the coordinated management of multiple resources, such as the CPU, network, and disk, (2) end-to-end (E2E) real-time guarantees across the use of multiple resources, and (3) feedback control across multiple resources. None of these properties is supported as a first-class feature within the state-of-the-art real-time operating systems, but are left out as an inconvenient detail to be managed by DRE application programmers. In this paper, we shed light on this fundamental problem and make the case for greater research into the development of theory and a runtime systems for coordinated allocation and scheduling of multiple resources in real-time operating systems. We also present the outlines of our proposed solution approach, called the Multiple Resource Allocation and Scheduling (*MURALS*) framework, that aims to bridge this gap between the need for E2E timing requirements and the techniques to coordinate the use of multiple resources.

1. INTRODUCTION

Distributed real-time embedded (DRE) systems are key components of critical infrastructure including surveillance, target tracking, electric grid management, traffic control and safety, process control, robotics, avionics, communication systems, and even real-time networked games. DRE systems in these applications are required to use multiple heterogeneous resources, such the CPU, network bandwidth, main memory, and secondary storage, The heterogeneity of resources and their interactions calls for coordinated management across these resources to meet end-to-end (E2E) deadlines. While real-time scheduling for a single resource in isolation has been studied extensively, relatively little work has been done for *integrated allocation and scheduling of multiple heterogeneous resources* to meet E2E timing constraints. Absence of coordination across multiple resources can lead to failure in meeting E2E timing guarantees in critical DRE systems.

An example DRE application is a surveillance network [37, 49], which consists of a group of cameras (and other sensors) connected over an area of interest. Each camera periodically captures a video frame, which is compressed by an embedded processor and transmitted over a wired or wireless LAN to a command and control (C2) center. A C2 server receives compressed video frames from multiple cameras across the LAN and executes several concurrent activities, such as to monitor the battlefield or traffic status. The server decompresses the video frames, displays the video on monitors, processes each frame for surveillance purposes, triggers alarms for security or safety reasons if necessary, and logs the data to a storage device. In this example, an E2E real-time task is associated with an E2E deadline to support the required application QoS. It also consists of multiple distributed subtasks using different system resources. A subtask depends upon the successful and timely completion of the previous subtask(s) in the sequence, forming precedence constraints. To summarize, the following characteristics of DRE applications emerge: (1) Multiple heterogeneous resources are used; (2) The resource usages within an application are ordered forming a precedence graph; and (3) Execution of a repetitive sequence of correlated subtasks is bounded by an E2E deadline. Our focus is on the applications with the above characteristics.

In this paper, we make the case for greater research into the development of theory and runtime systems for coordinated allocation and scheduling of multiple resources. We discuss four key open research problems and possible solution strategies.

(1) Deadline Partitioning Techniques: Given a DRE task that requires the use of multiple resources to meet its E2E deadline, one needs a deadline partitioning algorithm during admission control and resource allocation that apportions the the E2E delay budget among the subtasks of the DRE task. Careful deadline partitioning is important because it determines the load on each individual resource. In particular, tighter delay budget at a resource can lead to higher resource load, resulting in fewer admissible E2E DRE tasks in future. We investigate algorithms to efficiently partition E2E deadlines among multiple underlying resources. The goal is to increase the success ratio, i.e., the fraction of submitted DRE tasks that are admitted and completed within their E2E deadlines. The key idea is to reduce the extent of load imbalance among different resources during

deadline assignment to prevent formation of resource bottlenecks. Although deadline assignments in multiprocessor systems have previously been studied [51], most existing research considers only a single isolated resource.

(2) Coordinated Runtime Scheduling of Multiple Resources: While an effective deadline partitioning algorithm is necessary to assign a delay budget to each subtask of an E2E task, it is not sufficient by itself to guarantee that the E2E deadline will be met. During task execution, one needs explicit coordination across runtime schedulers of different resources to ensure that each subtask is scheduled to complete before its assigned sub-deadline. This is not the case in traditional RTOSs where scheduling decisions at one resource are made oblivious of the scheduling decisions at other resources. It thus becomes the DRE application writer's responsibility to manage any cross resource timing dependencies among subtasks. We illustrate this problem and suggest possible approaches to address such scheduling dependencies at runtime.

(3) Statistical Performance Guarantees: Reserving resources for worst-case load requirements may lead to resource under-utilization in the common case of low offered load. Additionally, a number of DRE applications, such as visual tracking and traffic monitoring, can adapt to a small probability of violations in their E2E guarantees. In this light, the multiple resource allocation techniques could potentially exploit the statistical multiplexing nature of the resource usage among concurrent DRE tasks to improve the system's overall resource utilization efficiency. Thus one needs statistical multi-resource allocation algorithms, such as online measurement-based techniques, that can exploit the statistical multiplexing nature of the resource usage and distinct tolerance levels to QoS violations, to reduce overall resource requirements of DRE applications. We investigate the role of resource allocation algorithms that exploit statistical multiplexing effects across multiple resources not just along the traditional 'bandwidth' dimension, but also along an orthogonal 'delay' dimension. We outline algorithms that can support tasks with *distinct probabilistic delay guarantees*, i.e., if certain tasks can tolerate more delay violations, they can reserve less resources than the other tasks tolerating fewer violations.

(4) Feedback Control Across Multiple Resources. Workloads may dynamically vary in DRE applications. For example, the image processing frequency and compression ratio may change depending on the presence or absence of objects indicating security breaches or traffic jams. Further, resource could be overbooked due to statistical multiplexing. As a result, E2E deadlines can be missed. Thus one requires control theoretic techniques to support E2E timing guarantees across multiple resources in unpredictable environments. In general, statistical approaches can provide high-level resource usage monitoring and QoS management, while control theoretic approaches can support fine-grained QoS management to ensure that, for example, no more than 1% of E2E deadlines are missed in average, no more than 1.5% of deadlines are missed even when the system is in a transient state, and a transient miss ratio overshoot, if any, decays within the specified settling time. Statistical and control theoretic approaches for QoS management have been studied separately; however, interactions between them have rarely been investigated [66]. We outline possible approaches for applying a combination of statistical and control theoretic techniques in multiple resource environments.

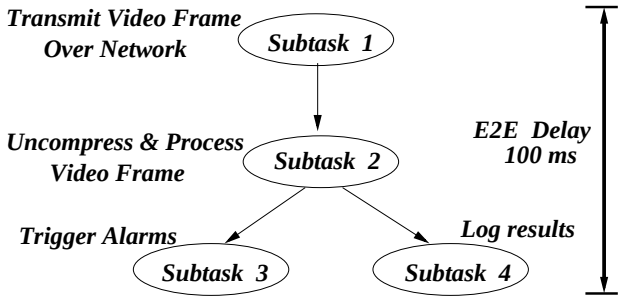

Figure 1: A task precedence graph for video surveillance with an E2E deadline.

We also present the outlines of our proposed solution approach, called the *Multiple Resource Allocation and Scheduling* (*MURALS*) framework and describe the ongoing development a proof-of-concept *MURALS* testbed on top of an commodity RTOS. The testbed includes not only kernel-level coordinated resource allocation mechanisms but also declarative APIs for E2E QoS specification. The API allows DRE application programmers to specify E2E tasks, their deadlines, periods, and precedence constraints across multiple resources. This information is utilized by kernel-level measurement-based QoS mapping scheme to automatically derive the low-level resource requirements from high-level performance requirements. Despite its importance, very little prior work has been conducted to provide declarative APIs and kernel support for deadline partitioning, coordinated scheduling, statistical guarantees, and feedback control. The goal of this paper is to make a case for greater research into this increasingly important subject.

2. DEADLINE PARTITIONING

A typical DRE task executes a set of subtasks related to each other via precedence constraints. For instance, Figure 1 shows the precedence graph of a video surveillance task executing four subtasks every period. Subtask 1 corresponds to the periodic compression and transmission of a video frame from a remote camera to a C2 server. At the C2 server, subtask 2 decompresses and processes the frame for target tracking, subtask 3 triggers alarms if necessary, and subtask 4 logs the results to a storage device in real-time. In this example, subtasks 3 and 4 can proceed concurrently once subtask 2 completes.

A precedence graph only describes the partial ordering but not the timing relationships among subtasks. For instance, the surveillance application may need to perform the E2E processing of each video frame within 100ms. This application-level performance requirement imposes a timing constraint for the E2E task and its subtasks in the precedence graph. Guaranteeing application-level QoS requires more than just local real-time scheduling for each individual resource, because it can only guarantee the subtask level QoS. For the DRE task shown in Figure 1, subtask 1 must be completed early enough to leave time for subtasks 2, 3, and 4 to complete before the E2E deadline.

A key problem illustrated in this example is *how to partition the E2E task deadline to meet the timing and precedence constraints, while improving the overall efficiency of resource utilization and E2E success ratio.* This requires specific algorithms that assign intermediate sub-deadlines

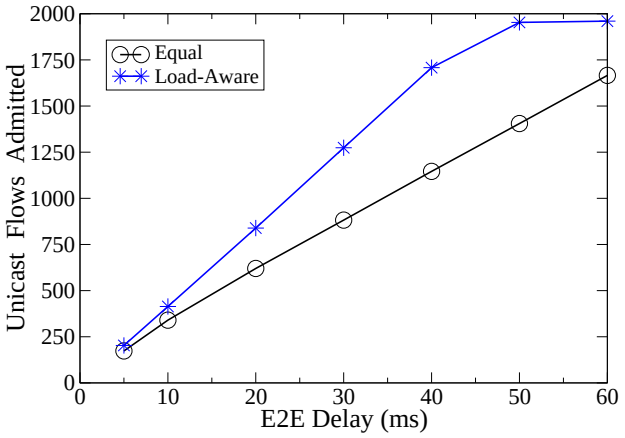

Figure 2: Flows admitted vs. E2E delay bound over Sprint IP Backbone. Hops=6. Flow data rate=100kbps. Link speeds:45–200Mbps.

for each subtask, while accounting for current and future load at each resource. Assigning a sub-deadline to a subtask also entails a specification of the load on the subtask's corresponding resource; in general, a tighter sub-deadline implies a higher resource load. Therefore, partitioning the delay budget, i.e., the E2E deadline, opens up an opportunity for load balancing across multiple resources resulting in efficient resource utilization. Rate-based schedulers, such as Virtual Clock [65] or WFQ [43], permit explicit mapping between latency bound requirements and bandwidth reservations [64]. Thus, the queuing delay experienced by a subtask at any resource scheduler is inversely proportional to the bandwidth reserved on its behalf. If a subtask needs a smaller delay budget, the corresponding resource reservation has to be larger, which imposes a heavier load on the resource. It is possible that some resources in the system may be more heavily loaded than others. Thus, one could partition the E2E deadline in such a manner that the more loaded resources are assigned a larger proportion of the E2E delay budget. This ensures that critical resources do not deplete long before less critical resources, thus preventing system-wide bottlenecks.

In the previous surveillance example, suppose the E2E deadline is 120ms. Consider that a UAV (Unmanned Aerial Vehicle) or traffic sensing node can capture and compress a fixed size video frame in 20ms using the dedicated embedded processor, whereas the network transmission delay and processing delay at the C2 server are variable as these resources are shared with other surveillance nodes. Thus, the remaining delay budget of 100ms needs to be partitioned among the network link and the C2 server's CPU. Assume that the C2 server's CPU is already 20% utilized and network link is 80% utilized. Instead of equally partitioning the delay budget between the network link and the CPU, a more sensible partition can be to assign 20ms to the CPU and 80ms to the network in proportion to their respective loads, and still meet the E2E deadlines.

This approach has shown promising initial results [17, 18] in which our delay partitioning techniques significantly reduce the load imbalance across multiple resources. In [17], we proposed a load-aware delay partitioning approach for sequential precedence graphs. We showed that one can increase the number of tasks admitted into the system by assigning delay budget D_i to each subtask i as per Equation 1 below, such that the E2E delay bound $D \geq \sum_i D_i$.

$$D_i = M_i + \frac{k_i M_i}{\sum_{j=1}^{m} k_j M_j} S \ , \quad k_i = \sqrt{\frac{W_{avg_i}}{W_i}} \qquad (1)$$

Here M_i is the minimum delay budget required to complete the subtask i, W_i is the amount of work such as the number of CPU cycles or bytes read/written, S is the slack in delay budget given by $S = D - \sum_i M_i$, and W_{avg_i} is the average amount of work requested by a DRE task at resource i. In [18], we proposed an iterative algorithm for delay partitioning along multi-hop network paths where each network link corresponds to one resource. Figure 2 shows that our load-aware delay partitioning algorithm admits up to 39% more network flows compared to equal partitioning, under the same network setup and input workload.

There are several open research challenges in the deadline partitioning problem.

Resource Specific Overheads: In practice, an exact algorithm for deadline partitioning across different resources needs to consider hard-to-characterize overheads such as context switching, network transmission overheads, and disk seek/rotational latency. Unfortunately, a large body of real-time scheduling theory ignores these overheads. Thus, a set of tasks may not be perfectly schedulable across multiple resources even if the available capacity is theoretically sufficient. Thus there is a need for a global admission control mechanism, which first consults each local resource scheduler to determine the minimum delay it can support using its current residual capacity. The delay partitioning algorithm could then apportion any remaining slack in E2E delay budget in a stepwise manner to decrease the demand on each resource. In each incremental step, the algorithm again needs to consult the local resource schedulers to determine how much the local delay will be increased due to the increase in resource demand and whether the tasks would still be schedulable. Such resource-aware algorithms for apportioning slack depend upon accurate mapping functions between the required delay bound and resource reservation for multiple resources, possibly distributed across a LAN.

Revising Earlier Allocations: There also exists a scope for algorithms that can make better deadline partitioning decisions, given additional knowledge of workloads. To leave as much resources unreserved as possible for the future use, our previous online admission control policy only considers a single DRE task for admission upon its arrival without changing the current reservations. The virtue of this approach is its simplicity and low overhead. However, if the delay partitioning algorithm has the flexibility of revising the previously assigned delay budgets of all the DRE tasks admitted earlier, then it can allocate *all* system resources to the current set of DRE tasks without leaving any unreserved resources. Thus one can design better resource allocation algorithms that can revise earlier allocation decisions, if necessary, to improve the efficiency of multi-resource utilization.

Optimizing for Power Efficiency: Another interesting open research problem is to exploit E2E delay partitioning to improve the power usage efficiency, instead of load balancing. Note that a shorter delay budget can translate to higher load and hence greater power dissipation at each resource. However, the mapping from delay budget to energy consumed for any given resource is likely to be a piecewise step rather than a continuous function. There is an opportunity to investigate different forms of such re-

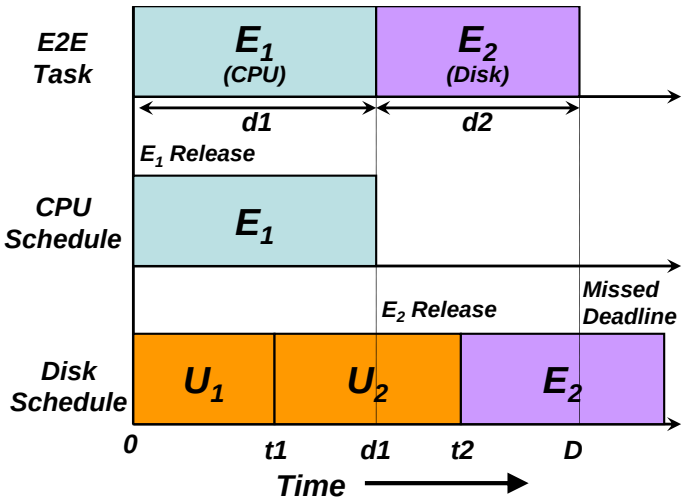

Figure 3: The inter-scheduler coordination problem at runtime for an E2E task with two subtasks, E_1 and E_2. The disk subtask E_2 misses the E2E deadline D after waiting for another non-preemptible I/O request U_2.

source mapping functions and their impact on delay partitioning strategies that optimize the energy consumption at low-power resources. A useful metric to gauge both timeliness and power usage efficiency is the *effective power consumption* = $total\_power\_consumption/success\_ratio$, lower value indicating greater effectiveness of delay partitioning algorithm. Existing work on Dynamic Slack Reclamation [34, 24] could be a starting point in this direction.

3. COORDINATED SCHEDULING

In the previous section, we discussed how effective partitioning of E2E delay budget among subtasks across multiple resources is essential to meet deadlines as well as to maintain high resource usage efficiency. However deadline partitioning constitutes only one component of the complete multi-resource allocation and scheduling framework. The other essential component is coordinated runtime scheduling of subtasks of an E2E task across multiple resources, in the absence of which even most judiciously partitioned deadlines might be missed.

To illustrate the problem, consider a simple example of an E2E real-time task E in Figure 3 that requires two resources – CPU and disk – to meet its E2E deadline of D. (We use disk I/O for illustration, though this example can easily be generalized for another I/O resource, such as network, or even for multiple I/O resources.) Assume that E2E delay budget D is partitioned as $d1$ and $d2$ across its CPU and disk subtasks, E_1 and E_2 respectively, where $d1 + d2 = D$. Both CPU and disk have their own independent real-time schedulers that, in the absence of any runtime coordination, maintain their own independent backlog queues of computation and I/O requests respectively. Assume that the subtask E_1 begins execution at CPU resource at time $t = 0$. In the meantime, the disk scheduler continues servicing other I/O requests U_1 and U_2 that are unrelated to the E2E real-time task E. At time $t1 < d1$, I/O request U_1 completes and U_2 is scheduled by the disk scheduler. At time $d1$, CPU subtask E1 completes and submits the real-time I/O task E_2 to the disk scheduler's runtime queue. However, even if E_2 is the most urgent I/O task at time $d1$, the disk scheduler needs to wait till time $t2$, when U_2 will complete service, before E_2 can be dispatched to the disk. This additional wait time could potentially cause E_2 to miss its I/O deadline, and consequently the E2E deadline, D.

Although the example above makes several simplifying assumptions about operating system behaviour, it serves to illustrate several fundamental problems when servicing a sequence of inter-dependent real-time subtasks across multiple resource schedulers.

Inter-Scheduler Coordination: Meeting the E2E deadline D depends upon the timely execution of component subtasks at *multiple* resource schedulers – here CPU and disk. In the absence of runtime coordination, each resource scheduler makes its own independent and locally optimal scheduling decisions because it is unaware of precedence constraints that control the release times of different real-time subtasks. For example, before time $d1$ in Figure 3, the disk scheduler was independently scheduling tasks U_1 and U_2, unaware of the fact that a real-time task E_2 was about to arrive with a tight deadline.

I/O Request Non-preemptibility: In general, I/O requests tend to be non-preemptible, once issued. Consequently, when dealing with multi-resource E2E tasks, a higher priority (static or dynamic) real-time I/O subtask may arrive at the I/O scheduling queue only to wait for the completion of a non-preemptible lower priority I/O request that's already in progress. Loosely speaking, this results in a short-term "priority inversion" that can potentially cause the higher priority real-time subtask to miss its E2E deadline.

Execution Time Prediction: While execution time of CPU subtasks can be predicted with reasonable accuracy, either through static code analysis or runtime profiling, the execution times for I/O subtasks are not as predictable. For example, disk response depends highly upon the seek and rotational latency whereas network response may depend upon switch congestion or channel access contention in broadcast media. In a multi-resource context, unpredictability in execution time might not only affect the timeliness the single I/O subtask in question, but also the timeliness of other dependent subtasks in the sequence of an E2E DRE task.

Advance Notification of Subtasks: None of the existing real-time operating systems address the above issues within a framework of coordinated multi-resource runtime scheduling, and it is undoubtedly a challenging research problem. To start making the problem manageable, our *MURALS* system incorporates an inter-scheduler coordination mechanism for multi-resource E2E real-time tasks that works as follows. Whenever a real-time E2E task begins execution, information about expected future arrival time and deadline of each of its subtasks are sent to the corresponding local resource schedulers. In the earlier example, at the start time 0, not only does the subtask E_1 begin execution at the CPU, but the disk scheduler is also given the information that subtask E_2, with delay budget of $d2$, will be ready at time $d1$. This enables the disk scheduler to insert an empty placeholder I/O request for task E_2 in its scheduling queue. At the scheduling time instant $t1$, the disk scheduler now has additional information to decide whether request U_2 can start and finish service early enough that the future real-time request E_2 can meet still its deadline. Admittedly, the

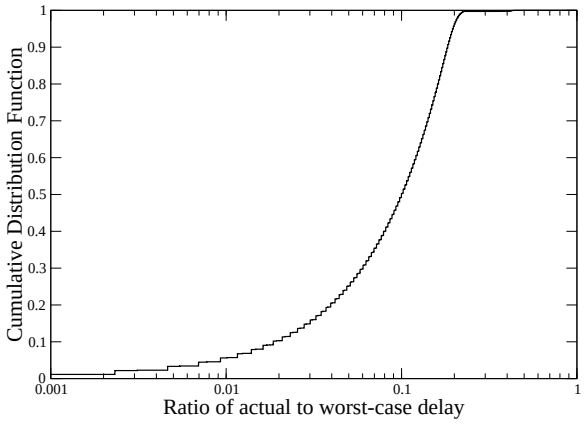

Figure 4: The CDF of the ratio of the actual delay experienced to the worst case delay expected by VoIP packets sharing a 10Mbps link.

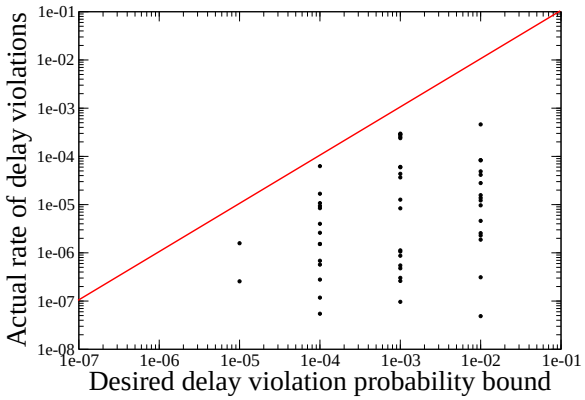

Figure 5: Effectiveness of DDM in differentiating between network flows with different tolerance to delay violations. Flows that have lower tolerance to violations experience fewer deadline misses. Each data point corresponds to one network flow. Delay bound=20ms. Link capacity = 10Mbps.

advance notification to the disk scheduler may not provide a foolproof guarantee that E_2 will always meet its deadline (for instance a long non-preemptible I/O task that began before time instant 0 might yet delay E_2). Nor does advance notification address the inherent unpredictability of I/O completion times. However this additional inter-scheduler coordination does help the local resource schedulers make better informed scheduling decisions from timeliness standpoint at a larger number of scheduling instances than without the coordination. For example, one option that the disk scheduler can exercise at time $t1$ is to follow a non-work-conserving policy by not scheduling any request between $t1$ and $d1$, thus keeping the disk idle until the subtask E_2 becomes eligible for service. This enables the disk scheduler to sacrifice throughput for the sake of timeliness when necessary – an option it cannot exercise without the advance information about E_2.

4. STATISTICAL DELAY GUARANTEES

Another challenge faced in DRE systems using multiple resources is that of *resource under-utilization*. While reserving resources for the peak load ensures that individual DRE tasks always meet their performance targets under all conditions, it ignores the reality that individual resources do not encounter peak load situations in the common case. Additionally, a number of DRE applications, e.g., visual tracking and traffic monitoring, can adapt to a small probability of violations in their QoS guarantees. Thus, the multiple resource allocation techniques could exploit the statistical multiplexing nature of the resource usage among DRE tasks to improve the system's resource utilization efficiency.

Traditional approaches tend to exploit statistical multiplexing along the *bandwidth dimension*, where the offered load per resource is lower than the reserved bandwidth share, due to which one can oversubscribe the underlying resource. Additionally, one can also leverage statistical multiplexing along the **delay dimension**. The latter effect arises from the fact that not all real-time applications will generate their peak request bursts at the same time. For instance, among a set of ON-OFF Voice over IP (VoIP) flows traversing a shared network link, it is unlikely that all VoIP flows will be in their ON state simultaneously. The consequence of this multiplexing is that actual delays rarely approach worst-case delay bounds that are based on all real-time applications generating their peak burst simultaneously. Note that statistical multiplexing in the delay dimension is orthogonal to that along the bandwidth dimension.

Consider Figure 4 showing the cumulative distribution function (CDF) $Prob(r)$ of the ratio of the actual to worst case delay experienced by various network packets (subtasks) at a network access link (resource). The distribution $Prob(r)$ gives the probability P that the actual delay D encountered by a request is smaller than r times its worst-case delay D^{wc} where $0 \leq r \leq 1$. Figure 4 shows that most requests experience less than 1/4th of their worst-case delays. In general, given a subtask that requires a latency bound of D with a violation probability bound of P, the following relationship holds from the definition of CDF: $D = D^{wc} \times Prob^{-1}(1-P)$. The nature of the worst-case delay term D^{wc} depends upon the nature of the specific local resource scheduler and the amount of resource bandwidth ρ reserved by the subtask. For instance, if the resource is scheduled using the Weighted Fair Queuing scheduler [43] then $D^{wc} = \frac{\sigma + L_{max}}{\rho} + \frac{L_{max}}{C}$, where σ is the maximum request burst size, L_{max} is the maximum request size and C is the total resource capacity. Hence, given the measured CDF $Prob(r)$, we can derive the required resource reservation ρ that satisfies the application latency requirements (D, P).

Our earlier work [19, 20] has successfully demonstrated techniques to exploit statistical multiplexing in the context of individual network and storage resource allocations. Figure 5 shows one example of the effectiveness of our approach using *delay distribution measurements* (DDM) in differentiating between the QoS requirements of multiple network flows that share a link and have different tolerance levels to delay bound violations. DDM has been shown to improve the resource usage efficiency of network and storage resources by up to a factor of 3 in the presence of statistical QoS constraints.

While statistical multiplexing in the context of single resource in isolation has been studied extensively [28, 29, 47, 7, 27, 6, 26, 45, 8, 56, 59], there has been little work in the direction of exploiting **statistical multiplexing across**

multiple heterogeneous resources. Consider a multi-resource DRE task, as in Figure 1, which requires an E2E delay bound of D with a violation probability bound of P. *How does one partition the E2E statistical QoS requirement (D,P) into individual subtasks requirements* (D_i, P_i) *such that resource loads can be balanced and the number of admitted DRE tasks can be increased as much as possible?* We outline a few possible solutions below.

Simple Partitioning Problem: A straightforward approach is to find a partition such that $\sum_i^n D_i \leq D$ and $\prod_i^n (1 - P_i) \geq (1 - P)$ for a set of subtasks consisting an E2E DRE task. We are currently investigating an iterative algorithm for this version of the partitioning problem, which is similar in structure to the basic delay partitioning algorithms discussed in Section 2. In every iteration, we can first estimate a delay partition assuming a fixed probability partition and then find a new probability partition using the fixed delay partition calculated in the previous step. This process continues till the delay and probability partitions obtained from two consecutive iterations are within a predefined threshold of each other.

General Partitioning Problem: Note that, in the above version of the partitioning problem, the condition on partitioning E2E violation probability is more conservative than necessary. In particular, it assumes that the entire DRE task can satisfy its E2E violation probability bound only if each subtask satisfies its local violation probability bounds. In contrast, while a subtask i could violate its local sub-deadline at one resource, its next subtask $i + 1$ in sequence could complete well ahead of its sub-deadline, thus making up for the lost time, and still meet the E2E deadline. Thus an open research problem is to model this general partitioning problem taking into account inter-dependencies among the current service loads at each resource, potentially yielding significant gains.

5. FEEDBACK CONTROL

Resource requirements of E2E tasks may dynamically vary in DRE applications, for example, due to varying image processing time or propagation delay. The relative importance of incoming data flows may change in time. For instance, certain data flows may actually capture enemy aircraft or traffic accidents, while the others do not deliver important data. In this case, the DRE application can increase the frequency of more important E2E tasks. Also, possible overbooking due to statistical multiplexing can overload the system. Given dynamic workloads, it is necessary to design *control theoretic techniques* to manage the miss ratio of E2E deadlines.

Figure 6 illustrates a possible architecture for feedback control at a C2 system. First, incoming packets are classified. Packet processing is scheduled at local resource schedulers via deadline partitioning and statistical multiplexing algorithms. Note that these algorithms may not be precise as workloads can vary dynamically, causing deadline misses under overload. The performance monitor measures the E2E miss ratio at every sampling instant, e.g., 1s, and informs the feedback controller of the current miss ratio m_c. At the k^{th} sampling period, the controller computes the error $e(k) = m_s - m_c(k)$ where m_s is the miss ratio set-point, e.g., 1%. Based on $e(k)$, it computes the control signal $\delta r(k)$. When $\delta r(k) < 0$, remote sensing nodes, e.g., UAVs or traffic monitoring nodes, are required to reduce the aggregate

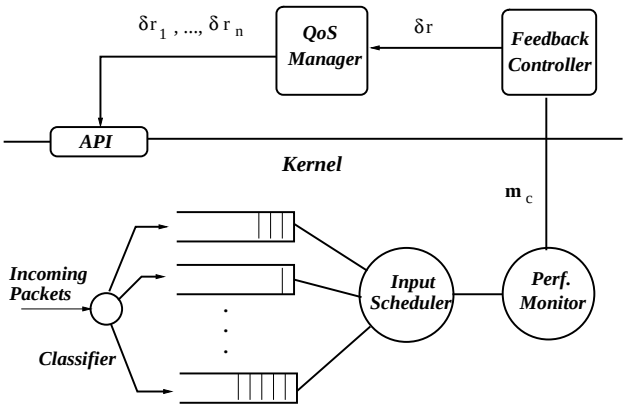

Figure 6: E2E Deadline Control Architecture

sensing and transmission rate by $|\delta r(k)|$. The QoS manager aware of application semantics or the instrumented DRE application itself divides $\delta r(k)$ among the incoming traffic flows to ensure that the most important flow, e.g., the flow tracking the largest number of targets, receives the highest rate. The resulting rates $\delta r_1, \delta r_2, ..., \delta r_n$ for n incoming sensor data flows are forwarded to a resource allocator, which accordingly reallocates resources between the n flows and sends the new required sensing rates to the remote sensor nodes, if necessary, to meet the QoS requirements. Below we describe some open research problems in developing an E2E timing control architecture.

Handling Input Queue Backlogs: A possible backlog in the incoming packet queues shown in Figure 6 can adversely affect the overall control performance. The system performance may not change immediately for a new control signal when there is a backlog to which the previous delay budget is already distributed [66]. Thus, there could be dead-time in control, which can result in a nonlinear system behavior [44]. One possible approach is redistributing the delay budget of the jobs already in the queues for the cost of performing resource re-allocation. In such situations, the resource allocation algorithms need to be lightweight so that one can sporadically re-allocate resources to improve real-time performance without affecting E2E tasks. We envision linearly approximating the system model via the relation between the aggregate packet arrival rate and E2E miss ratio. The miss ratio $m(k)$ at the k^{th} sampling period can be modeled by an n^{th} $(n \geq 1)$ order difference equation with unknown coefficients $\{a_i, b_i | 1 \leq i \leq n\}$ initialized to zero:

$$m(k) = \sum_{i=1}^{n} a_i m(k-i) + \sum_{i=1}^{n} b_i r(k-i) \qquad (2)$$

where $r(k-i)$ and $m(k-i)$ are the aggregate packet arrival rate and E2E deadline miss ratio at the $(k-i)^{th}$ sampling instant, respectively. Eq 2 denotes that the current miss ratio is dependent on the packet arrival rates and miss ratios measured at the previous sampling periods. System identification (SYSID) techniques [44, 42] can be applied to identify the model parameters in Eq 2. At the same time, there is a need to experimentally determine the specific order of Eq 2 producing high-accuracy SYSID results verified by standard techniques such as root mean square errors [44].

Self-Tuning Based Techniques: If the linear time invariant (LTI) control based on the system model and SYSID above is not applicable, e.g., due to a large backlog or wire-

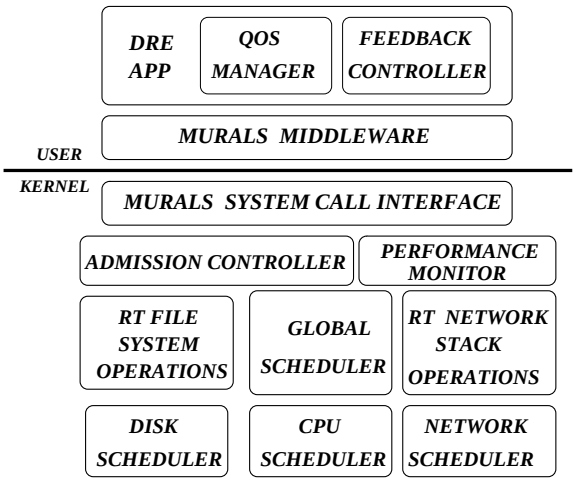

Figure 7: The architecture of the *MURALS* framework.

less communication jitters in a noisy environment, it is also feasible to consider either a *self-tuning regulator* (STR) [53] or *self-tuning fuzzy controller* (STFC) [10]. A STR performs online SYSID to find the model coefficients in Eq 2 and tune the controller parameters online. In this way, the controller can tune itself if the system dynamics change in time. Fuzzy control is useful when the underlying system is hard to model mathematically. A linguistic rulebase instead of a mathematical system model can be designed for feedback control. For example, if the current overshoot is high and it is diverging from the set-point, the rulebase can generate a negative large control signal. On the other hand, if an overshoot shows a self-decaying pattern compared to the previous one, the rulebase generates a negative small signal for control stability. Especially, a STFC can tune itself, via a separate rulebase for self-tuning, to improve the performance considering the current system behavior. In real-time systems, the performance of several LTI models are compared [3], but other models are not compared extensively. Thus there is a need to undertake in-depth comparisons of the performance and complexity of the different control models, i.e., LTI and more advanced models, in the context of multi-resource DRE systems. As a result, control models achieving good performance with low complexity can be developed. Other key issues to be investigated along with control modeling include sampling period selection and stability analysis.

6. MURALS TESTBED

We are developing a prototype *MURALS* testbed on top of the TimeSys Linux [54] real-time operating system. The goal of *MURALS* is not to develop another comprehensive RTOS, but to demonstrate the effectiveness of core algorithms and techniques required for coordinated multiple resource allocation, with substantial rethinking of fundamental APIs and system architecture.

Figure 7 shows the architecture of the *MURALS* framework. Real-time applications access the *MURALS* API that interacts with the kernel subsystem using a set of *MURALS*-specific system calls. An admission controller makes admission decisions and performs deadline partitioning at the time a new DRE task registers itself. A performance monitor constantly tracks the timing behavior of E2E tasks to inform the feedback controller of the E2E miss ratio at every sampling instant.

The core of the *MURALS* kernel is organized around a two-level scheduling architecture. It consists of a *global scheduler*, which is aware of each application's task graph and estimated resource requirements, and a set of *local schedulers*, each of which correspond to a particular resource. The global scheduler ensures that a subtask's dependencies are all satisfied before it is executed by the the corresponding local resource scheduler. Local real-time schedulers manage individual resources and make scheduling decisions based on both the subtask deadlines and resource utilization efficiency. In this way, the global scheduler acts as a glue between local resource schedulers using its global knowledge of E2E deadlines and estimated resource requirements of each E2E task. Individual system resources employ rate-based real-time scheduling algorithms. CPU scheduler is a variant of Virtual Clock [65]. Disk scheduler is a rate-based variant of the our work-conserving DSSCAN algorithm [16]. Real-time network access is guaranteed by the wired and wireless variants of our Real-Time Ethernet (RETHER) protocol [50, 57]. These algorithms are modular and replaceable by any other rate-based schedulers.

The *MURALS* API library allows DRE application programmers to declaratively specify individual resource requirements and E2E timing constraints. The API facilitates creation and execution of a precedence graph, its component subtasks, and their dependencies. The library in turn informs the *MURALS* module in the kernel of the application requirements. A DRE application creates an initially empty precedence graph by first invoking the `Create_graph` function as `Graph_id = Create_graph(Deadline,Tolerance)` to specify that the precedence graph for an E2E task needs to be executed within the specified deadline and tolerance to delay violations. The application can possibly spawn many such precedence graphs that execute concurrently. The precedence graph can be populated by application programmers with individual subtasks such as read, write, or computation. For example, a write subtask can be added to the graph via `Register_write` call as `Subtask1 = Register_write( Graph_id, File_descriptor, Buffer, Size)` where the file descriptor can represent either regular files or network sockets. Thus, the subtask can transmit data across the network or write data to the corresponding file. A read subtask can be registered in a similar fashion. Also, a computation subtask can be registered as `Subtask2 = Register_compute( Graph_id, Compute_func)`. Dependencies among subtasks can be specified in a pairwise manner. For example, `Depend(Subtask1, Subtask2, Graph_id)` specifies that Subtask2 can execute only after Subtask1 completes. The subtask dependencies are conveyed by the API to the *MURALS* kernel module verifying that the graph remains acyclic.

The registration of a subtask does not execute the subtask immediately. Rather, it informs the kernel *how* to execute this subtask *when* the kernel is asked to do so. This separation of *how* to execute a subtask from *when* to execute it is a departure from conventional operating system designs. It can provide greater flexibility in real-time application programming and resource scheduling. The DRE task can then be repeatedly executed by invoking `Exec_graph(Graph_id)`, which transparently executes all the subtasks according to precedence constraints within the specified E2E deadline.

7. RELATED WORK

A vast amount of research work has been conducted in real-time resource allocation and scheduling from different perspectives. We discuss the most relevant research results.

Multiple Resource Coordination: The body of work on real-time multi-resource coordination is relatively sparse, with none of the techniques supported in state-of-the-art RTOSs. The continuous media resource (CM-resource) model [4] is a framework meant for continuous media, e.g., digital audio and video, applications. Clients make resource reservations for the worst-case workload. The meta-scheduler coordinates with the CPU scheduler, network, and file-system to negotiate delay guarantees and the required buffer size on behalf of clients. Xu et. al. [63] present simulation results for a multi-resource reservation algorithm that determines the E2E QoS level for an application under resource availability constraints. The work on Cooperative Scheduling Server (CSS) [48] performs admission control for disk I/O requests by reserving both the raw disk bandwidth and CPU bandwidth required for processing disk requests. Timing constraints are partitioned into multiple stages and each of them is guaranteed to complete before its deadline on a particular resource. However, the deadline is partitioned based on a fixed slack sharing scheme rather than considering the resource utilization efficiency, possibly causing load imbalance across different resources. Q-RAM [31, 46, 13, 14] considers the problem of allocating multiple resources in one or more QoS dimensions to maximize the overall system utility. Spring Kernel [52] provides real-time support for multiprocessor and distributed systems using dynamic planning based scheduling. Real-time applications are written using Spring-C and resource requirements are specified using System Description Language. [33] models the effect of Linux network device driver on the schedulability analysis of real-time applications. Deadline partitioning has also been studied in the context of CPU resources alone for multiprocessor systems [51], for CPU and disk resource [17], and multi-link network paths [18].

Real-time Operating Systems: Numerous RTOSs support real-time scheduling for independent system resources, but lack a coordinated multi-resource allocation and scheduling mechanism to support E2E delay bounds. Some of these are Real-time Mach [55], Linux/RK [35], TimeSys Linux [54], RT-Linux [12], KURT Linux [30], QLinux [21], Eclipse/BSD [5], Rialto [25], and Nemesis [32].

Statistical Multiplexing: While statistical multiplexing has been extensively studied in relation to networks and to some extent in cluster-based services, relatively little attention has been paid towards statistical multiplexing effects in multi-resource real-time systems. Knightly and Shroff [28] provide an excellent overview of admission control approaches for link-level statistical QoS. Kurose [29] derived shared probabilistic bounds on delay and buffer occupancy using the concept of stochastic ordering for network nodes that use FIFO scheduling. Several analytical approaches [27, 9, 22, 47] have also considered multiplexing with shared buffers in single and multiple link settings. The notion of Effective Bandwidth, introduced by Kelly [26], is an important measure of bandwidth resource usage by flows relative to their peak and mean usage. A comparative study [8] of several MBAC algorithms [45, 23, 11, 15] under FIFO service concluded that none of them accurately achieve loss targets. Urgaonkar et. al. [56] perform resource overbooking via offline capacity profiling in shared hosting platforms for CPU and network resources. Vin et.al [59] explored statistical admission control algorithms for media servers. Vernick et.al [58] reported empirical measurements from implementations of statistical admission control algorithms in a fully operational disk-array video server.

Feedback Control: Most classical, open-loop real-time scheduling algorithms assume precise *a priori* knowledge of workloads. Feedback control has recently been applied to real-time performance management, because it does not require accurate system models for performance guarantees. A survey of feedback control for QoS management is provided in [2]. FC-UM [39, 40] provides the specified miss ratio and utilization in a single processor environment. DEUCON [61] manages the E2E CPU utilization in a multiprocessor environment via predictive model control. CAMRIT [60] applies control theoretic techniques to support the timeliness of image transmissions across one wireless link. All these approaches only consider a single resource, i.e., the CPU or transmission rate. HiDRA [49] manages both CPU and network bandwidth utilization via feedback control for target tracking. AQuoSA[36] proposes a control theoretic strategies to dynamically adapt CPU reservations in the Linux kernel. Control theoretic techniques have also been applied to manage the performance of a web server [1, 62, 38] and web cache [41]. However little research has been directed towards applying control theoretic techniques for real-time multi-resource applications.

8. CONCLUSION

Modern real-time systems increasingly consist of applications that need to use multiple heterogeneous resources to complete critical tasks within bounded end-to-end (E2E) delay. This argues for coordinated allocation and scheduling of multiple resources, such as the CPU, network, and disk, in real-time operating systems (RTOS). Unfortunately, state-of-the-art RTOS do not support multi-resource coordination as a fundamental construct in the system design. This is presumably because the complex inter-resource interactions are poorly understood when it comes to resource allocation and runtime scheduling decisions at each resource in highly dynamic and distributed real-time systems. We examined some of the fundamental problems in this area and made the case for greater research effort into the development of theory and a runtime systems for coordinated allocation and scheduling of multiple resources in real-time operating systems. We discussed four open research problems and possible solution strategies in the areas of E2E deadline partitioning, explicit coordination across resource schedulers, statistical performance guarantees, and feedback control across multiple resources. We also described our current research efforts in the design and implementation of a *MURALS* testbed that addresses the above research problems. Even though we may not have covered all the major research issues in multi-resource allocation and scheduling, our paper has sought to identify and motivate investigation into some of the fundamental problems in this increasingly important area.

9. REFERENCES

[1] T. F. Abdelzaher and N. Bhatti. Adaptive Content Delivery for Web Server QoS. In *IWQoS*, 1999.

[2] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback Performance Control in Software Services. *IEEE Control Systems*, 23(3), 2003.

[3] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. H. Son. Experimental Evaluation of Linear Time-Invariant Models for Feedback Performance Control in Real-Time System. *Real-Time Systems*.

[4] D.P. Anderson. Metascheduling for continuous media. *ACM Trans. on Computer Systems*, 11(3):226–252, Aug. 1993.

[5] J. Blanquer, J. Bruno, E. Gabber, M. Mcshea, B. Ozden, and A. Silberschatz. Resource management for QoS in Eclipse/BSD. In *Proc. of FreeBSD'99 Conf., Berkeley, CA, USA*, Oct. 1999.

[6] R. Boorstyn, A. Burchard, J. Leibeherr, and C. Oottamakorn. Statistical service assurances for traffic scheduling algorithms. *IEEE JSAC*, 18(13):2651–2664, Dec. 2000.

[7] J.-Y Le Boudec and M. Vojnovic. Stochastic analysis of some expedited forwarding networks. In *IEEE Infocom'02*, June 2002.

[8] L. Breslau, S. Jamin, and S. Shenker. Comments on performance of measurement-based admission control algorithms. In *Proc. of IEEE INFOCOM*, 2000.

[9] R.L. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Trans. on Information Theory*, 37(1):114–131, 1991.

[10] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An Introduction to Fuzzy Control*. Springer, 1996.

[11] S. Floyd. *Comments on measurement-based admission control for controlled load services*. Tech. Rep., Lawrence Berkeley Labs, 1996.

[12] FSMLabs. Rtlinux free. http://www.rtlinuxfree.com/.

[13] Sourav Ghosh, Ragunathan Rajkumar, Jeff Hansen, and John Lehoczky. Scalable Resource Allocation for Multi-Processor QoS Optimization. In *ICDCS*, 2003.

[14] Sourav Ghosh, Ragunathan Rajkumar, Jeffery Hansen, and John Lehoczky. Integrated Resource Management and Scheduling with Multi-Resource Constraints. In *RTSS*, 2004.

[15] R. Gibbens and F. Kelly. Measurement-based connection admission control. In *Proc. of 15th Intl. Teletraffic Conference*, June 1997.

[16] K. Gopalan and T. Chiueh. *Real-time Disk Scheduling Using Deadline Sensitive Scan*. Technical Report ECSL-TR-92, Experimental Computer Systems Lab, Stony Brook University, January 2001.

[17] K. Gopalan and T. Chiueh. Multi-resource allocation and scheduling for periodic soft real-time applications. In *Multimedia Computing and Networking*, Jan. 2002.

[18] K. Gopalan, T. Chiueh, and Y.J. Lin. Delay budget partitioning to maximize network resource usage efficiency. In *Proc. IEEE INFOCOM'04, Hong Kong, China*, March 2004.

[19] K. Gopalan, T. Chiueh, and Y.J. Lin. Probabilistic delay guarantees using delay distribution measurements. In *Proc. of ACM Multimedia 2004, New York, NY*, October 2004.

[20] K. Gopalan, L. Huang, G. Peng, T. Chiueh, and Y.J. Lin. Statistical admission control using delay distribution measurements. *ACM Trans. on Multimedia Computing, Commn, and Apps*, 2(4), Nov. 2006.

[21] P. Goyal, X. Guo, and H. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *SOSP*, Oct. 1996.

[22] F. M. Guillemin, N. Likhanov, R. R. Mazumdar, and C. Rosenberg. Extremal traffic and bounds for the mean delay of multiplexed regulated traffic streams. In *IEEE INFOCOM'02*, June 2002.

[23] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. *IEEE/ACM Transactions on Networking*, 5(1):56–70, Feb. 1997.

[24] R. Jejurikar and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Design Automation Conf.*, June 2005.

[25] M. B. Jones, D. Rosu, and M. Rosu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proc. of SOSP*, pages 198–211, Oct. 1997.

[26] F. Kelly. Notes on effective bandwidths. *In Stochastic Networks: Theory and Applications*, 4:141–168, 1996.

[27] G. Kesidis and T. Konstantopoulos. Worst-case performance of a buffer with independent shaped arrival processes. *IEEE Communication Letters*, 4(1):26–28, Jan. 2000.

[28] E.W. Knightly and N. B. Shroff. Admission control for statistical QoS. *IEEE Network*, 13(2):20–29, March 1999.

[29] J. Kurose. On computing per-session performance bounds in high-speed multi-hop computer networks. In *Proc. of ACM Sigmetrics'92*.

[30] KURT-Linux. http://www.ittc.ku.edu/kurt/.

[31] C. Lee, J.P. Lehoczky, D.P. Siewiorek, R. Rajkumar, and J.P. Hansen. A scalable solution to the multi-resource QoS problem. In *IEEE RTSS'99*.

[32] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE JSAC*, 14(7):1280–1297, Sept. 1996.

[33] M. Lewandowski, M. Stanovich, T. Baker, K. Gopalan, and A. Wang. Modeling device driver effects in real-time schedulability analysis: Study of a network driver. In *RTAS, Bellevue, WA*, April 2007.

[34] C. Lin and S.A. Brandt. Improving soft real-time performance through better slack reclaiming. In *Proc. of Real-Time Systems Symposium (RTSS)*, 2005.

[35] Linux/RK. http://www.cs.cmu.edu/∼rajkumar/linux-rk.html.

[36] G. Lipari, L. Abeni, T. Cucinotta, L. Marzario, and L. Palopoli. Qos management through adaptive reservations. *Real-Time Systems*, 29, 2005.

[37] J. Loyall, R. Schantz, D. Corman, J. Paunicka, and S. Fernandez. A Distributed Real-Time Embedded Application for Surveillance, Detection, and Tracking of Time Critical Targets. In *IEEE RTAS*, 2005.

[38] C. Lu, Y. Lu, T.F. Abdelzaher, J.A. Stankovic, and

S.H. Son. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE Transactions on Parallel and Distributed Systems*, 17(9), Sept. 2006.

[39] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Real-Time Systems*, 23(1/2), May 2002.

[40] C. Lu, X. Wang, and C. Gill. Feedback Control Real-Time Scheduling in ORB Middleware. In *the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[41] Ying Lu, Tarek F. Abdelzaher, and Avneesh Saxena. Design, Implementation, and Evaluation of Differentiated Caching Services. *IEEE Transactions on Parallel and Distributed Systems*, 15(4), May 2004.

[42] J. Oh and K. D. Kang. An Approach for Real-Time Database Modeling and Performance Management. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2007.

[43] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE Trans. on Networking*, 1(3):344–357, Jun 1993.

[44] C. L. Phillips and H. T. Nagle. *Digital Control System Analysis and Design (3rd edition)*. Prentice Hall, 1995.

[45] J. Qiu and E. Knightly. Measurement-based admission control with aggregate traffic envelopes. *IEEE/ACM Transactions on Networking*, 9(2):199–210, April 2001.

[46] R. Rajkumar, C. Lee, J.P. Lehoczky, and D. P. Siewiorek. Practical solutions for QoS-based resource allocation. In *IEEE RTSS*, Dec. 1998.

[47] M. Reisslein, K.W. Ross, and S. Rajagopal. A framework for guaranteeing statistical QoS. *IEEE/ACM Transactions on Networking*, 10(1):27–42, February 2002.

[48] S. Saewong and R. Rajkumar. Cooperative scheduling of multiple resources. In *IEEE RTSS'99*, pages 90–101, Dec. 1999.

[49] N. Shankaran, X. Koutsoukos, D. Schmidt, Y. Xue, and C. Lu. Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems. In *Euromicro Conference on Real-Time Systems*, 2006.

[50] S. Sharma, K. Gopalan, N. Zhu, G. Peng, P. De, and T. Chiueh. Implementation Experiences of Bandwidth Guarantee on a Wireless LAN. In *Multimedia Computing and Networking*, Jan 2002.

[51] J. W. S.Liu. *Real-Time Systems*. Prentice Hall, 2000.

[52] J. Stankovic and K. Ramamritham. The Spring Kernel: A new paradigm for real-time systems. *IEEE Software*, 8(3), May 1991.

[53] K. J. Åström and B. Wittenmark. *Adaptive Control*. Prentice Hall, 1995.

[54] Timesys Inc. *http://www.timesys.com/*.

[55] H. Tokuda, T. Nakajima, and P. Rao. Real-time mach: Toward a predictable real-time system. In *USENIX Mach Workshop*, 1990.

[56] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *OSDI*, Dec. 2002.

[57] C. Venkatramani and T. Chiueh. Design, implementation, and evaluation of a software-driven real-time ethernet protocol. In *SIGCOMM'95*.

[58] M. Vernick, C. Venkatramani, and T. Chiueh. Adventures in building the stony brook video server. In *ACM Multimedia*, Nov 1996.

[59] H. M. Vin, P. Goyal, , and A. Goyal. A statistical admission control algorithm for multimedia servers. In *ACM Multimedia'94*.

[60] X. Wang, H.-M. Huang, V. Subramonian, C. Lu, and C. Gill. CAMRIT: Control-based Adaptive Middleware for Real-time Image Transmission. In *RTAS*, 2004.

[61] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems. *IEEE Trans. on Parallel and Distributed Systems*, 18(7):996–1009, July 2007.

[62] J. Wei and C-Z. Xu. eQoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers. *IEEE Transactions on Computers*, 55(12), Dec. 2006.

[63] D. Xu, K. Nahrstedt, A. Viswanathan, and D. Wichadakul. QoS and contention-aware multi-resource reservation. In *HPDC*, Aug. 2000.

[64] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proc. of IEEE*, volume 83(10), pages 1374–1396,, 1995.

[65] L. Zhang. Virtual Clock: A new traffic control algorithm for packet switching networks. In *Proc. of ACM SIGCOMM'90, Philadelphia, PA, USA*, pages 19–29, Sept. 1990.

[66] R. Zhang, S. Parekh, Y. Diao, M. Surendra, T. Abdelzaher, and J. Stankovic. Control of Weighted Fair Queueing: Modeling, Implementation, and Experiences. In *Intl. Symposium on Integrated Network Management*, 2005.