QoS-based Management of Multiple Shared Resource in Dynamic Real-Time Systems

Klaus Ecker, Frank Drews, and Jens Lichtenberg
School of EECS, Ohio University, Athens, OH 45701
{ecker, drews, lichtenj}@ohio.edu

Abstract

Dynamic real-time systems require adaptive resource management to accommodate varying processing needs. We address the problem of resource management with multiple shared resources for soft real-time systems consisting of tasks that have discrete QoS settings that correspond to varying resource usage and varying utility. Given an amount of available resource, the problem is to provide on-line control of the tasks' QoS settings so as to optimize the overall system utility. We propose several heuristic algorithms that will be shown to be compatible with the requirements imposed by our control theoretical resource management framework: (1) By only making incremental adjustments to QoS settings as available resources change, they provide low run-time complexity, making them suitable for use in on-line resource managers (2) Differences between actual utility and optimal utility do not accumulate over time, so there is no long-term degradation in performance. (3) The lower and upper bound on actual utility can be calculated dynamically based on current system conditions, and absolute bounds can be calculated statically in advance. (4) It is possible to respond to the actual resource possible, allowing all resources to be used and tolerating misspecification of task resource requirements.

1 Introduction

Traditional approaches to the design and development of real-time systems use worst-case execution times of tasks and provably correct analysis techniques to guarantee that critical system tasks will always complete prior to their deadlines. The most common approach is rate monotonic analysis [1] which uses bounds on processor utilization to guarantee that real-time periodic tasks always meet their deadlines. These traditional (and often pessimistic) approaches to the design and development of real-time systems may result in systems that are under-utilized for much of the operational time the system, resulting in poor quality of service. A number of factors may lead to this poor performance, including overly pessimistic worst-case execution times, environmental conditions, infrequent critical instants [1], inaccurate resource profiles, etc. The discrete event controller that we describe controls system resource usage by modifying quality of service parameters so that the overall system benefit is maximized. When the system resource being controlled is processor utilization, our approach is directly applicable to real-time systems since total processor utilization provides a schedulability test, i.e., a guarantee that real-time tasks will meet their deadlines whenever the processor utilization is less than or equal to some value. Since the discrete controller cannot guarantee that the resource usage is bounded, our approach is not applicable to hard-real systems where deadline can never be missed. However, our approach has many advantages in soft real-time systems where deadlines may be missed occasionally. The principle motivation behind this work is the feedback control of soft real-time systems (multimedia, certain feedback control systems) where the quality of service to the user can be improved if the amount of resources provided to the individual tasks is increased. An example of such as system is the RT-Phone example provided in the papers associated with Q-RAM [2, 3, 4, 5]. Our approach differs from that of Q-RAM since we make resource reallocations at run-time using feedback control.

This paper addresses the problem of resource management with task sharing multiple resources for soft real-time systems (no hard deadline requirements) consisting of tasks that have discrete QoS (Quality of Service) settings that correspond to varying resource usage and varying benefit to the end user. We are starting with a system model that similar to the one used in Q-RAM [2, 3, 4, 5]. Given an amount of available resource, the problem is to provide on-line control of the tasks' QoS settings so as to optimize the overall system benefit. Since the complexity of the problem precludes optimal solutions, we present two heuristic algorithms with the following properties.

- They have low run-time complexity, making them suitable to be employed as an online algorithm in dynamically changing environments.
- Unlike Q-RAM, our approach makes incremental adjustments to QoS settings as available resources change, avoiding the calculation time that would be incurred by recalculating all the QoS settings.
- Differences between actual utility and optimal do not accumulate over time, so there is no long-term degradation in performance. We derive upper and lower bounds on actual utility. They can be calcu-

1-4244-0054-6/06/$20.00

lated dynamically based on current system conditions, and an absolute bound can be calculated statically in advance.

- The algorithms will be employed in and have to be compatible with our feedback control based resource management architecture [27] that allows response to actual resource availability, allowing all resources to be used and tolerating misspecification of task resource requirements.

Section 2 presents the problem. Section 3 introduces the notation, a formalized model and, based on this, formulates the problem formally. Section 4 presents discusses the complexity status of the problem, exact algorithms, and introduces the fast heuristic algorithms Discrete Kuhn-Tucker, Improve_Allocation, Steepest Ascent. Performance bounds are derived for some of these algorithms. Some simulation results verifying the performance bounds are added in Section 5. Section 6 deals with the problem of dynamic resource availability changes, and discusses fast heuristic adaptation strategies. Performance verification by simulation is presented in Section 7.

2 Related Work

The problem of allocating resources to real-time applications has been studied in literature from different angles. Several authors have addressed resource allocation for real-time systems with QoS constraints. Burns et al [6] define a task model where each task is given a preference at run-time and a set of service alternatives that can be used to complete the task. QoS-based dynamic resource management tries to guarantee the feasibility of real-time systems as well as let the system produce maximum benefit, but does not explicitly address problems that arise from environmental run-time variations. Humphrey et. al. [7] propose the DQM architecture, which allows choosing operating levels to control the resource requirements for an application. The DQM monitors the performance of applications and interacts with the operating system in order to change the application to a lower operating level. The DQM is intended for use with soft real-time multimedia applications, so adaptation may occur over several periods. However, DQM uses a worst case execution time analysis to determine application resource usage. DQM does not provide a mathematical optimization model, and does not guarantee that optimal, or even near-optimal, choices have been made. In the QuO [8, 9, 10, 11] framework, applications adjust their own service levels to improve performance and react to the environment on their own accord, so there is no way to globally optimize the set of choices made for all applications. Nor is QuO able to guarantee real-time deadlines. In Q-RAM [2,3,4,5], an algorithmic approach is developed to find an allocation of tasks to resources such that the system can satisfy some quality of service requirements as well as produce maximum benefit. However, the authors do not explicitly discuss the use of Q-RAM for QoS optimization in dynamic environments. The application of control-theoretic methods to the design of real-time systems has recently met with considerable success. Common challenges in real-time system design such as nonlinear and stochastic plant models, effector limitations, unknown disturbances, and noisy sensor data identified in [13] indicate a strong connection with control theory and applications. A series of works [14,15,16,17,18] address performance specifications, mathematical modeling, controller design, and performance analysis for scheduling problems in soft real-time systems. Their feedback control architecture is realized in middleware called ControlWare and its effectiveness for quality of service control is demonstrated in a web server environment. Limitations of linear systems and control methods are discussed in [19] with remedies presented that draw from scheduling and queuing theory. Related studies involving a variety of real-time system applications, performance objectives, mathematical modeling approaches, and feedback control architectures can be found in [20, 21, 22, 23, 24, 25, 26]. A common thread through much of this work is that quality of service attributes of soft real-time tasks are adjusted via feedback control based on on-line system measurements. The underlying performance objective is then to optimize an aggregate quality of service metric while adhering to resource constraints and coping with an uncertain dynamic environment.

3 Problem Formulation and Notation

The system in consideration has a number of hard and soft real-time tasks. Tasks require certain resources. The resources required by h.r.-t. tasks depend on the work loads defined by the environment. It is generally assumed that there are sufficiently many resources to process the hard r.-t. tasks within the range of the possible workloads. During run-time, as long as the work loads are below their limits, there will be some of the resources unused and can hence be allocated to the soft real time tasks. We assume in our model that the utility gained from the execution of a soft r.-t. task depends on the amount of resources allocated to the task. The objective then is to maximize the total utility by a proper allocation of the surplus resources.

In our model we distinguish $m \geq 1$ different resource types $\{R_1,\ldots, R_m\}$, each of them available in several units. Let r_i^{max} be the number of units of resource type R_i available at current time. For each resource type, the units are to be distributed among the soft r.-t. tasks. $M := (r_1^{max},\ldots, r_m^{max})$ denotes the vector of current resource limitations.

We assume that there is given a set of periodic soft real-time tasks $T = \{T_1, \ldots, T_n\}$. The processing times and periods may depend on the amount of allocated resources. A resource allocation defines how many resources of each type are assigned to the tasks. The resources currently assigned to a task T_j are denoted by $r(T_j) := (r_{j1}, \ldots, r_{jm})$ with $0 \leq r_{ji} \leq r_i^{max}$ for $i = 1, \ldots, m$.

Introducing the set of resource vectors $\mathcal{G} = \{(r_1,\ldots, r_m) \mid r_i \in \{0,\ldots, r_i^{max}\}, i = 1,\ldots, m\}$, also referred to as resource space, allows us to define an allocation as a mapping $alloc : T \rightarrow \mathcal{G}$. An allocation *alloc* is feasible if $\sum alloc(T_j)$ is component-wise not larger than $(r_1^{max}, \ldots, r_m^{max})$. For convenience reason we define the zero allocation as the allocation $zero(T_j) = (0,\ldots,0)$ for $j = 1,\ldots, n$.

The performance of a task T_j is measured by a discrete and concave utility function u_j that depends on the type and number of assigned resources, $u_j : \mathcal{G} \rightarrow \mathbb{R}^{\geq 0}$ The utility functions we think of here are concave. Concave utility functions are widely used in this area

Examples of utility measures are:

- Inverse of processing time: The more resources a task has, the shorter the processing time, and the higher the benefit.
- Precision of computation: more resources lead to better computational results.

The total utility of an allocation *alloc* is defined by a function that aggregates the utilities of all tasks, possibly with priorities: $U(alloc) := \sum \pi_j u_j (alloc(T_j))$ where π_j is the priority of task T_j (π_j can of course be included in the definition of u_j).

Objective: Given a resource limitation $M := (r_1^{max},\ldots, r_m^{max})$, find an allocation with maximum utility.

Example: $m = 2$, $M = (10, 15)$; u_1 is specified on the values $u_1(0) = 0$, $u_1(1) = 12$, $u_1(2) = 18$, etc., and $u_2(0) = 5$, $u_2(1) = 8$, $u_2(2) = 10$, etc. . The objective is to find a point $(a_{j1},\ldots, a_{jm}) \in G$ for each function u_j such that $\sum_{j=1,\ldots,n} \{ u_j(r_{j1},\ldots, r_{jm}) \}$ is maximum, subject to $\sum_{j=1,\ldots,n} r_{ji} \leq r_i^{max}$ for $i = 1, \ldots, m$.

4 Optimization Problem

Algorithms for Maximizing Utility

In the case of continuous and single-dimensional concave functions u_j the Kuhn-Tucker theorem can be applied to find a solution with maximum total utility. This algorithm can easily be adapted to the discrete functions u_j, where it follows a steepest ascent rule. We refer to this algorithm as the *Discrete_Kuhn_Tucker* (*DKT*) algorithm.

If there is more than one resource type, an allocation that maximizes utility can be found by scanning the resource space $\mathcal{G}$. In the PhD thesis of Chen Lee [5] a linear programming algorithm is presented for the case of n tasks and two resource types, that runs in $O(n\cdot(r_1^{max} r_2^{max})^2)$ time. Despite the low time complexity, the space complexity is of the same order, which seems to be unrealistic high for practical problems with large values of r_i^{max}.

Since high system adaptability requires very short reaction times, we are interested in fast heuristic approaches that are able to find reasonable solutions. One heuristic is based on a discrete adaptation of the Kuhn-Tucker algorithm [ref]. In algorithm $DKT(R_i)$, the units of resource R_i are allocated step by step such that maximum utility increase is gained. If there is only one resource type this strategy is known to be optimal [2] .

A simple generalization of *DKT* is the *Multidimensional_Discrete_Kuhn_Tucker* algorithm, which applies $DKT(R_i)$ to the resource directions in some chosen order, say $R_1, \ldots, R_m$.

Algorithm *Multidimensional_Discrete_Kuhn_Tucker* (*MDKT*).

```
for i := 1 to m do DKT(R_i);
```

The time complexity of *MDKT* is $O(n(r_1^{max}+\ldots+ r_m^{max}))$. It has, however, to be expected that the utility of the resulting allocation will not be the optimum, and particularly will depend on the order in which the resource types are chosen. In Section 5 some performance measures can be found.

An alternative heuristics being worth analyzed is *Steepest_Ascent*. As *DKT* it starts from the allocation *zero*, and, at each step, allocates one resource unit such that the total utility U is increased by the maximum possible amount.

Algorithm *Steepest_Ascent* (*STAS*)

```
repeat
  choose a task T_j and an available resource R_i such
     that r_ji is maximum;
  assign one unit of R_i to T_j;
  until all resource units are allocated;
```

Time complexity of *STAS* is $O(n(r_1^{max}+\ldots+ r_m^{max}))$, which is of the same order as for *MDKT*.

In the next sections we analyze the performance of the algorithms *MDKT* and *SA* by deriving lower and upper bounds for the total utility (4.2), and by experimental studies (5).

Estimating lower and upper bounds

A lower bound for the maximum utility can easily be determined by assigning all resource units to a task that has a largest maximum utility,

$$\max\{ u_j(r_1^{max}, \ldots, r_m^{max} \mid j = 1, \ldots, n \}.$$

$\sum r_i^{max}$ is an obvious upper bound for the total utility. As long as we do not know more about the functions u_j , this bound cannot be improved, because by choos-

ing each function u_j sufficiently close (but still concave) to the constant r_i^{max}, the utility may become arbitrarily close to this bound.

For deriving better bounds one has to take more particularities of the input functions into consideration. Our idea is to replace the given utility functions by special ones for which the optimum can be computed with low computational complexity. First we need a few preparations.

There is a special case for which optimality of *MDKT* can be proved. Assume that, for each task T_j, u_j has the property that, for all $i \in \{1, \ldots, m\}$,

$$u_{ji}(r_i) := u_j(r_1,\ldots,r_{i+1},\ldots,r_m) - u_j(r_1,\ldots,r_i,\ldots,r_m) \quad (1)$$

depends only on the value of r_i, i.e., this difference is independent of r_k for $k \neq i$ (see figure 1).

Definition. Given a concave function $u : \mathcal{G} \to \mathbb{R}^{\geq 0}$, let $pr_j^{a_i \in 0..r_i^{max}}(u)$ be the projection of u where the values of the components r_i, $i \neq j$ are for fixed. We call $u : \mathcal{G} \to$

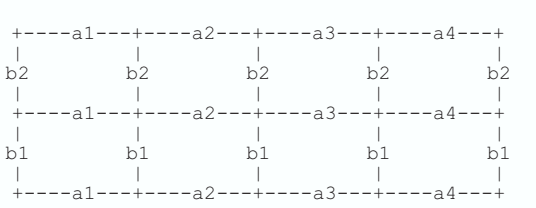

Figure 1 Illustration of a special case of a utility function; a_i and b_i are the slopes

$\mathbb{R}^{\geq 0}$ *uniform* if the projections $pr_j^{a_i \in 0..r_i^{max}}(u)$ do not depend on the particular choice of the a_i, i.e., if $pr_j^{a_i \in 0..r_i^{max}}(u) = pr_j^{a_i' \in 0..r_i^{max}}(u)$ for all a_i, a_i' and all $i \neq j$.

For uniform concave functions we have the following theorem.

Theorem. *The algorithm MDKT is optimal for uniform utility functions*.

Proof. When applying $DKT(R_i)$, the resulting utility does not depend on allocations of the other resources $R_k \neq R_i$. Hence two conclusions can be drawn:

(a) The result is independent of the order in which the $DKT(R_i)$ are executed.

(b) The resulting allocation is optimal. This can be followed by proof by contradiction. □

We make use of the following obvious property (denote by U_{ALG} the utility computed by a heuristic algorithm *ALG*).

Proposition. *Let U_{opt} be the maximum utility that can be obtained from tasks $T_1, \ldots, T_n$ with the given functions $u_1,\ldots, u_n$. Let furthermore $u_1', \ldots, u_n'$ be alternative utility functions that are smaller than $u_1, \ldots, u_n$, i.e., for all $j \in \{1,\ldots,n\}$ and for all $\vec{r} \in \mathcal{G}$, $u_j'(\vec{r}) \leq u_j(\vec{r})$; and let U_{opt}' be the corresponding optimal utility. Then $U_{opt}' \leq U_{opt}$, $U_{ALG}' \leq U_{ALG}$ for $ALG \in \{MDKT, STAS\}$.*

Proof. trivial. □

For finding *lower and upper bounds*, we replace the functions u_j by partially linear functions s_j with $s_j(\vec{r_j}) \leq u_j(\vec{r_j})$ (and h_j with $h_j(\vec{r_j}) \geq u_j(\vec{r_j})$, respectively) for all $\vec{r_j} = (r_{j1}, \ldots, r_{jm}) \in \mathcal{G}$ that are close to the original functions u_j.

We show an easy way to choose suitable functions s_j (for h_j this can be done correspondingly) is the following: Find first the minimum increase of utility u_j when one additional resource unit is allocated to T_j:

Determine the minimum and maximum differences $\delta_j(r_k)$ and $\Delta_j(r_k)$, respectively, when all resource values r_j except r_k are fixed,

$$\delta_j(r_k) = \min\{\, u_j(r_1,\ldots, r_m) - u_j(r_1',\ldots, r_m') \mid r_i = r_i' \in \{0,\ldots, r_i^{max}\} \text{ for } i \neq k \text{, and } r_k' = r_k + 1\}$$

$$\Delta_j(r_k) = \max\{\, u_j(r_1,\ldots, r_m) - u_j(r_1',\ldots, r_m') \mid r_i = r_i' \in \{0,\ldots, r_i^{max}\} \text{ for } i \neq k \,\}$$

$$(k = 1, \ldots, m).$$

Notice from the definition of s_j and h_j that both functions are uniform in the sense of the definition.

Defining the functions s_j and h_j by

$$s_j(r_1,\ldots, r_m) = \sum_{l=1}^{m} \sum_{k=0}^{r_l - 1} \delta_l(0,\ldots 0,k,0\ldots,0)\,, \quad (2)$$

and

$$h_j(r_1,\ldots, r_m) = \sum_{l=1}^{m} \sum_{k=0}^{r_l - 1} \Delta_l(0,\ldots 0,k,0\ldots,0)\,. \quad (3)$$

We see that both functions are uniform in the sense of the above definition. From the theorem we therefore know that *MDKT* is optimal on these functions.

Because of its low computational complexity, *MDKT* gives us a useful mean to derive lower and upper bounds for given instances.

Corollary. *For any given an instance (i.e. a set of $m > 0$ concave utility functions u_j),*

$$U_{MDKT}(s_1,\ldots, s_m) \leq U_{MDKT}(u_1,\ldots, u_m) \leq U_{MDKT}(h_1,\ldots, h_m)\,. \quad \square$$

For a given instance with functions u_j, the functions s_j and h_j defined by (2) and (3) allow quickly deriving lower and upper bounds.

5 Simulation Results

We performed elaborate simulations for the case of $m = 2$ resource types. Utility functions $u_i(x,y)$ were generated in two different ways:

(a) directly with using integer random numbers from the interval [0,100],

(b) by evaluating the function

$$u_i(x,y) = a_1(i)\ln(x+1)(1+a_2(i)/(x+a_3(i))) + a_4(i)\ln(y+1)(1+a_5(i)/(x+a_6(i))) \quad (4)$$

where the parameters were chosen arbitrarily between 0 and 200. Figure 2 shows the utility function

$z = 71.0*\log(x+1)*(1/(x+7.8)+1) + 121.0*\log(y+1)*(1/(x+5.7)+1).$

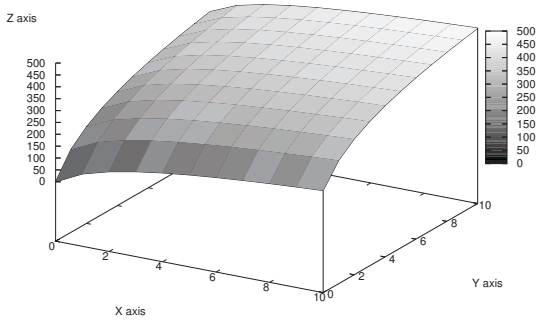

Figure 2 Example of a utility function.

Lower and upper bounds for U_{MDKT} are approx. 10 % off the optimum. Experimental studies also show that for functions defined by (4), *STAS* performed on average one order of magnitude better than *MDKT*. So experimentally we were able to verify that the upper and lower bounds of MDKT are also valid for STAS.

We performed simulations with fixed task numbers between 2 and 35, and resource units between 2 and 100. In each case we performed 10000 tests. Figure 3 compares the performance of *MDKT* and *STAS* against the optimum for task numbers varying from 5 to 35. We see that, in average, the utility of the *MDKT*-solutions is not more than 0.3 % below the optimum, with decreasing distance for larger task numbers. For *STAS*, the average utility is even closer to the optimum. We conjecture that generally these algoroitms lead to better (indeed almost optimal) results for larger numbers of tasks.

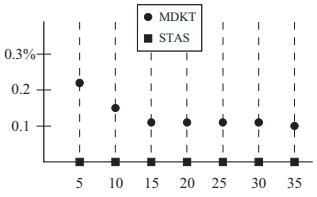

Figure 3 Deviation of *MDKT* and *STAS* from optimum.

The statistics presented in Figure 4 compares *MDKT* and *STAS* for 5 tasks and varying numbers of allocated resource units. There are 15 units of each resource available, and the number of resources each tasks may get is increasing; larger numbers lead to a higher competition. The figure shows that the higher the competition the better are is the performance of *MDKT*. Again there is no significant deviation of *STAS* from the optimum.

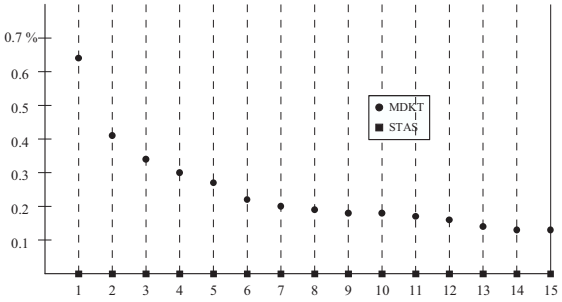

Figure 4 Higher competition (right) leads to better utility.

6 Dynamic Changes of Resource Availability

Finally we turn to the dynamic situation where resource limitations change over time. If resource units are withdrawn from soft real-time tasks then one or more tasks have to run at a lower utility, thus resulting in a smaller total system utility. If the number of available resource units increases the tasks should take advantage in order to increase their contribution to the total system utility. We propose the following simple and obvious rule of resource re-allocation:

General strategy:

- If one of the resource limits is reduced by one unit, reduce the assignment of this resource for a task that has minimum loss of utility.
- If one of the resource limits is increased by one unit, then assign the additional unit to a task that offers maximum utility gain.

First simulation studies already show that the system behaves stable. Further investigations are in preparation.

7 Conclusions

Extending the work of QRAM, we have presented heuristics to optimize utility with respect to QoS settings on multiple shared resources that meets the basic requirements of our control theoretical resource management framework. The heuristics have been shown to be efficient and stable control algorithms that optimize task QoS settings, within known margins, with respect to overall utility, that do not accumulate errors over time, and that tolerate task resource misspecification. We have shown experimentally that the QoS settings produced by the algorithm tend to be quite good; actual utility produced is typically very close to if not identical with the optimal. In future work we intend to extend the theory in the following ways: (1) Handle

utility functions with arbitrary discrete resource allocation steps (2) Handle hard real-time tasks, for single and multiple instances of a resource, and for multiple resource types. (3) Handle the problem of moving tasks from one resource instance to another to ensure schedulability and optimize utility.

References

[1] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[2] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A qos-based resource allocations model. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1997.

[3] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. Practical solutions for qos-based resource allocation problems. In *In Proceedings of the IEEE Real-Time Systems Symposium*, pages 315–326, 1998.

[4] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A scalable solution to the multi-resource QoS problem. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 315–326, 1999.

[5] C. Lee and D. Siewiorek. On quality of service optimization with discrete qos options. In *In Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium (RTAS '99)*, pages 276–286, 1999.

[6] A. Burns. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.

[7] M. Humphrey, S. Brandt, G. Nutt, and T. Berk. The DQM architecture: Middleware for application -centered QoS resource management. In *Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, pages . 97–104, 1997.

[8] D. Karr, C. Rodrigues, J. Loyall, and R. Schantz. Controlling quality-of-service in a distributed video application by an adaptive middleware framework. In *Proceedings of ACM Multimedia*, 2001.

[9] C. Koliver, K. Nahrstedt, J. Farines, J. Fraga, and S. Sandri. Specification, mapping and control for QoS adaptation. *Special issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):143–174, 2002.

[10] Y. Krishnamurth, V. Kachroo, D. Karr, C. Rodrigues, J. Loyall, R. Schantz, and D. Schmidt. Integration of QoS-enabled distributed object computing middleware for developing next-generation distributed applications. In *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems*, 2001.

[11] B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, and E. Huh. DynBench: A dynamic benchmark suite for distributed real-time systems. In *Proceedings of the Parallel and Distributed Processing: IPPS/SPDP Workshops*, pages 1335–1349, 1999.

[12] J. Loyall, P. Rubel, R. Schantz, M. Atighetchi, and J. Zinky. Emerging patterns in adaptive, distributed real-time, embedded middleware. In *Proceedings of the 9th Conference on Pattern Language of Programs*, 2002.

[13] J. Hellerstein. Challenges in control engineering of computing systems. In *Proceedings of the 2004 American Control Conference*, pages 1970 – 1979, 2004.

[14] J. Stankovic, C. Lu, S. Son, and G. Tao. The case for feedback control real-time scheduling. In *EuroMicro Conference on Real-Time Systems*, June 1999.

[15] C. Lu, J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 2000.

[16] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu. Feedback control scheduling in distributed real-time systems. In *IEEE Real-Time Systems Symposium*, 2001.

[17] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Special issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):85–126, 2002.

[18] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software systems. *IEEE Control Systems Magazine*, 23(3):74–90, June 2003.

[19] T. Abdelzaher, Y. Lu, R. Zhang, and D. Henriksson. Practical application of control theory to web services. In *Proceedings of the 2004 American Control Conference*, pages 1992 – 1997, 2004.

[20] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analyis of a reservation-based feedback scheduler. In *Proceeding of the Real-Time Systems Symposium*, 2002.

[21] G. Buttazzo and L. Abeni. Workload management through elastic scheduling. *Special Issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):7–24, July/September, 2002.

[22] A. Cervin, J. Eker, B. Bernhardsson, and K. Erzin. Feedback-feedforward scheduling of control tasks. *Special Issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):25–53, July/September, 2002.

[23] C. Koliver, K. Nahrstedt, J. Farines, J. Fraga, and S. Sandri. Specification, mapping and control for QoS adaptation. *Special issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):143–174, 2002.

[24] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Special issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):127–141, 2002.

[25] V. Vahia, A. Goel, J. Walpole, and M. Shor. Using dynamic optimization for control of real rate CPU resource management. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 6547 – 6552, 2003.

[26] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Design and evaluation of load control in web server systems. In *Proceedings of the 2004 American Control Conference*, pages 1980 – 1985, 2004.

[27] R. Judd, F. Drews, D. Lawrence, D. Juedes, B. Leal, J. Deshpande, and L. Welch, QoS-based Resource Allocation in Dynamic Real-Time Systems, *Proceedings of the 24th American Control Conference (ACC 2005), pp. 1745-1751, 2005.*