A Resource Allocation Model for QoS Management*

Ragunathan Rajkumar, Chen Lee, John Lehoczky†, Dan Siewiorek
Department of Computer Science
†Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213
{raj+, clee, dps}@cs.cmu.edu, †jpl@stat.cmu.edu

Abstract

Quality of service (QoS) has been receiving wide attention in recent years in many research communities including networking, multimedia systems, real-time systems and distributed systems. In large distributed systems such as those used in defense systems, on-demand service and inter-networked systems, applications contending for system resources must satisfy timing, reliability and security constraints as well as application-specific quality requirements. Allocating sufficient resources to different applications in order to satisfy various requirements is a fundamental problem in these situations. A basic yet flexible model for performance-driven resource allocations can therefore be useful in making appropriate tradeoffs.

In this paper, we present an analytical model for QoS management in systems which must satisfy application needs along multiple dimensions such as timeliness, reliable delivery schemes, cryptographic security and data quality. We refer to this model as Q-RAM (QoS-based Resource Allocation Model). The model assumes a system with multiple concurrent applications, each of which can operate at different levels of quality based on the system resources available to it. The goal of the model is to be able to allocate resources to the various applications such that the overall system utility *is maximized under the constraint that each application can meet its minimum needs. We identify resource profiles of applications which allow such decisions to be made efficiently and in real-time. We also identify application utility functions along different dimensions which are composable to form unique application requirement profiles. We use a video-conferencing system to illustrate the model.*

1 Introduction

1.1 Motivation

Many applications can provide better performance and quality of service given a larger share of system resources. For example, feedback control systems can provide better control with higher rates of sampling and control actuation. Multimedia systems using audio and video streams can provide better audio/video quality at higher resolution and very low latencies. Tracking applications can track objects at higher precision and accuracy if radar tracks are generated and processed at higher frequencies or if better, but more computationally intensive, algorithms are used. Real-time decision-making systems can receive, process and analyze larger amounts of data if more resources are made available. Interactive systems can provide excellent response times to users if more processing and I/O resources are made available.

Applications can therefore seek to improve the quality of delivered services if sufficient resources are available. For example, if encoding/decoding times were not significant, *all* transmitted data can be encrypted for security/privacy reasons. If spare resources were available, modules can be replicated to assure high availability of critical functionality. Conversely, when resources are tight, applications can still provide lower but acceptable behavior. For instance, a 30 frames/second video rate would be ideal for human viewing, but a smooth 12 fps video rate suffices under many conditions.

Given that applications can operate at high levels of quality or acceptably lower levels of quality based on the resources allocated to them, the following question arises: "How does one allocate resources to those applications when they run concurrently and contend for the same resource types?" This question of resource allocation is traditional in the sense that many papers in the domains of networking, real-time systems and distributed systems have attempted to answer it (e.g. [3]) in their own context. However, we are unaware of any significant work which allows requirements such as timeliness, security and reliable data delivery to be addressed and traded off against each other within the same context. Similarly, much of the QoS work focuses on allocating a single time-shared resource such as network bandwidth. In real-time systems, applications may need to have simultaneous access to multiple resources such as processing cycles, memory, network bandwidth and disk bandwidth, in order to satisfy their needs.

In this paper, we propose the QoS-based Resource Allocation Model (Q-RAM) as an *initial* step in addressing both of these problems:

*This work was supported in part by the Defense Advanced Research Projects Agency under agreements E30602-97-2-0287 and N66001-97-C-8527, and in part by the Office of Naval Research under agreement N00014-92-J-1524.

- satisfying simultaneous requirements along multiple QoS dimensions such as timeliness, cryptography, data quality and reliable packet delivery, and
- having access to multiple resources simultaneously.

We present resource allocation schemes to solve only the first problem dealing with multiple QoS dimensions. Resource allocation schemes in the presence of multiple resources are the subject of ongoing work and are beyond the scope of this paper.

1.2 Related Work

A significant amount of work has been carried out for making resource allocations to satisfy specific application-level requirements. Such work can be classified into various categories. The problem of allocating appropriate resource capacity to achieve a specific level of QoS for an application has been studied in various contexts. For example, [3] studies the problem of how to allocate network packet processing capacity assuming bursty traffic and finite buffers. In [2], the problem of the establishment of real-time communication channels is studied as an admission control problem. The Spring Kernel [15] uses on-line admission control to guarantee essential tasks upon arrival.

Various system-wide schemes have been studied to arbitrate resource allocation among contending applications. In [16], a distributed pool of processors is used to guarantee timeliness for real-time applications using admission control and load-sharing techniques. The Rialto operating system [5] presents a modular OS approach, the goal of which is to maximize the user's perceived utility of the system, instead of maximizing the performance of any particular application. No theoretical basis is provided to maximize system utility. A QoS manager is used in the RT-Mach operating system to allocate resources to application, each of which can operate at any resource allocation point within minimum and maximum thresholds [7]. Applications are ranked according to their semantic importance, and different adjustment policies are used to obtain or negotiate a particular resource allocation.

Once a resource allocation decision has been made, various scheduling schemes are available to ensure that the allocation decisions can be carried out. A CPU resource reservation scheme [11] is used in RT-Mach to guarantee and enforce access to an allocated resource once a resource allocation decision has been made. A large portion of real-time scheduling theory deals with this problem and uses fixed priority schemes [9, 8, 13, 6], dynamic priority schemes [1, 4] or heuristic schemes [17]. The basic requirements of a QoS model in high assurance applications are presented in [18]. It proposes that the QoS attributes of timeliness, precision and accuracy can be used for system specification, instrumentation and evaluation.

The Q-RAM model we propose can be considered to be a generalization of at least two models previously studied in the literature. First, the *imprecise computation* model proposed by Liu et al. [10] considered the problem of optimally allocating CPU cycles to applications which must satisfy minimum CPU requirements, but can produce better results with additional CPU cycles. The frequency of each application remains constant, while the computation time per instance of an application can be varied. The results were generally assumed to improve linearly with additional resources. Secondly, Seto et al. [14] have studied the problem of allocating CPU cycles optimally to feedback control applications whose control quality improves in concave fashion with higher frequencies of operation. The computation time per instance of an application remains constant.

Our proposed model can be viewed as a combination and broad generalization of these models. First, we allow either the computation time or the frequency of an application to vary. Secondly, and more importantly, we seek to generalize the resource allocation model to support multiple dimensions of quality (timeliness, data quality, reliable packet delivery, security achieved through cryptography, etc.) for each application to support the simultaneous allocation of multiple resource types (CPU and disk bandwidth, for example) for each application. The model in its general form only assumes that an application's quality will not decrease with any increase in resource allocation. We only deal with cryptographic security in this paper and the term 'security' will be used only in that sense.

The rest of this paper is organized as follows. In Section 2, we present our QoS-based Resource Allocation Model (Q-RAM) and illustrate the concepts behind the model using an actual video-conferencing system. In Section 3, we determine optimal resource allocation schemes for single variable QoS constraints. In Section 4, we identify the main considerations of multi-dimensional QoS problems and present optimal and greedy resource allocation for different cases. We also apply Q-RAM to the video-conferencing system and consider schedulability issues. In Section 5, we present our concluding remarks and discuss problems that remain unsolved.

2 Q-RAM: The QoS-based Resource Allocation Model

Q-RAM is based on a dynamic and adaptive application framework with the following characteristics:

- An application may need to satisfy many requirements: timeliness, security, data quality, dependability, etc.
- An application may require access to multiple resource types such as CPU, disk bandwidth, network bandwidth, memory, etc.
- An application requires a certain minimum resource allocation to perform acceptably. It may also improve its performance with larger resource allocations. This improvement in performance is measured by a utility function.

Q-RAM is a model in which resources can be allocated to individual applications with the goal of maximizing a global objective. The model is intended for use with static (off-line) allocation schemes, dynamic (on-line) allocation with admission control schemes, and 'timed' allocations where each resource allocation has a duration of validity associated with it.

2.1 Quality of Service Dimensions

Consider an application which obtains and transmits audio data. The application can use a reliable encoding scheme to tolerate and recover from bit errors during transmission. The data can be made secure by encrypting the transmitted packets. The application may process and transmit the audio data in smaller chunks to meet real-time constraints. The application may also choose to improve audio quality by increasing the size of each audio sample or by increasing its sampling rate. The application may also want to perform one or more these simultaneously but each option requires the use of additional resources. We refer to these quality aspects such as timeliness, reliability, security and data quality as *QoS dimensions*.

In Q-RAM, we consider a system in which multiple applications, each with its own set of requirements along multiple QoS dimensions, are contending for resources.

- Each application may have a minimum and/or a maximum need along each dimension.
- Each resource allocation adds some utility to the application and the system, with utility monotonically increasing with resource allocation.
- System resources are limited so that the maximal demands of all applications often cannot be satisfied simultaneously.

With the Q-RAM specifications, a resource allocation decision will be made for each application such that an overall system-level objective (called utility) is maximized.

2.2 The Definition of Q-RAM

Q-RAM is defined as follows. The system consists of n applications $\{\tau_1, \tau_2, \cdots, \tau_n\}$, $n \geq 1$, and m resources $\{\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_m\}$, $m \geq 1$. Each resource $\mathbf{R}_j$has a finite capacity and can be shared, either temporally or spatially. CPU and network bandwidth, for example, would be time-shared resources, while memory would be a spatially shared resource.

Let the portion of resource $\mathbf{R}_j$ allocated to application τ_i be denoted by $R_{i,j}$. We enforce $\sum_{i=1}^{n} R_{i,j} \leq \mathbf{R}_j$. Two issues need to be noted in the context of real-time systems in particular:

- *Utilization*: The resource allocation to an application will be in terms of the utilization of a resource. Once a certain utilization has been allocated, an application may either choose its own execution time and period to achieve that utilization or use an appropriate processor-sharing scheme such as weighted fair-sharing.
- *Schedulability*: The constraint $\sum_{i=1}^{n} R_{i,j} \leq \mathbf{R}_j$ implies that a resource can be "fully" consumed. As is well known, this is *not* always true for fixed-priority scheduling algorithms [9] but is true for the earliest deadline scheduling algorithm under ideal conditions. A different maximal resource constraint beyond the scope of this paper must be used to support fixed-priority schemes. For example, see [14].

We now introduce the following definitions:

- The *application utility*, U_i, of an application τ_i is defined to be the value that is accrued by the system when τ_i is allocated $\mathbf{R}^i = (R_{i,1}, R_{i,j}, \cdots, R_{i,m})$. In other words, $U_i = U_i(\mathbf{R}^i)$. U_i is referred to as the *utility function* of τ_i. This utility function defines a surface along which the application can operate based on the resources allocated to it.
- Each application τ_i has a relative importance specified by a weight w_i, $1 \leq i \leq n$.
- The *total system utility* $\mathbf{U}(\mathbf{R}^1, \cdots, \mathbf{R}^n)$ is defined to be the sum of the weighted application utility of the applications, i.e. $\mathbf{U}(\mathbf{R}^1, \cdots, \mathbf{R}^n) = \sum_{i=1}^{n} w_i U_i(\mathbf{R}^i)$.
- Each application τ_i needs to satisfy requirements along d QoS dimensions $\{Q_1, Q_2, \cdots, Q_d\}$, $d \geq 1$.
- The *dimensional resource utility* $U_{i,k} = U_{i,k}(\mathbf{R}^i)$ of an application τ_i is defined to be the value that is accrued by the system when τ_i is allocated $\mathbf{R}^i$ for use on QoS dimension $Q_k, 1 \leq k \leq d$.
- [1]An application, τ_i, has *minimal resource requirements on QoS dimension* Q_k. These minimal requirements are denoted by $R_i^{min_k} = \{R_{i,1}^{min_k}, R_{i,2}^{min_k}, \cdots, R_{i,m}^{min_k}\}$ where $R_{i,j}^{min_k} \geq 0, 0 \leq j \leq m$.
- An application, τ_i, is said to be *feasible* if it is allocated a minimum set of resources on every QoS dimension. We denote the total minimum requirements by $\mathbf{R}_i^{min} = \{R_{i,1}^{min}, R_{i,2}^{min}, \cdots, R_{i,m}^{min}\}$ where $R_{i,j}^{min} = \sum_{k=1}^{d} R_{i,j}^{\mathrm{min}_k}, 1 \leq j \leq m$.

In this paper, we assume that $m = 1$, i.e. only a single resource is being allocated.

2.3 Assumptions

We make the following assumptions:

A1. The applications are independent of one another.

A2. The available system resources are sufficient to meet the minimal resource requirements of each application on all QoS dimensions, $\mathbf{R}_i^{min}, 1 \leq i \leq n$.

A3. The utility functions U_i and $U_{i,k}$ are nondecreasing in each of their arguments. In some cases we will assume that these functions are concave and have two continuous derivatives.

[1]This aspect of the model is a simplification to be relaxed in future work. In general, multiple resource-tuples can yield a given QoS operating point.

A4. Each application, τ_i, has a weight w_i denoting its relative importance.

We make the following observations concerning these assumptions. First, if assumption **A1** does not hold, then the resource allocation methods still apply; however, the schedulability analysis needed to ensure that application timing requirements are met is more complicated. It must take into account phenomena such as the priority inversion that can occur with synchronization protocols.

Second, if assumption **A2** does not hold, then the minimal resource requirements cannot be met. If these requirements are not met, then some of the applications must be dropped. One can use a variety of techniques to determine which of the applications should be dropped, or one could even allow some applications to have less than their minimal resource allocations. Although this is a very important issue, it is beyond the scope of this paper.

Third, in view of **A4**, we can now define a *weighted utility function* for an application as $w_i * U_i$ and then solve the resource allocation problem for those weighted utility functions. Thus, one can remove the weights from the allocation problem. In our subsequent analysis, we use these weighted utilities and drop the weight function.

Note that U_i is not necessarily equal to $\sum_{j=1}^{m} U_{i,j}$. In other words, the utility obtained by an application τ_i from a resource $\mathbf{R}_j$ may not be additive with respect to its utility from another resource. This is because the application may need two or more resources *simultaneously* to achieve a certain utility. For example, an audio-conferencing application may need the CPU resource and the networking bandwidth resource in order to satisfy even a minimal QoS requirement.

2.4 The Objective

The objective of Q-RAM is to make resource allocations to each application such that the total system utility is maximized under the constraint that *every* application is feasible with respect to each QoS dimension. Stated formally, we need to determine

$\{R_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m\}$ such that $R_{i,j} \geq \sum_{k=1}^{d} R_{i,j}^{min_k}$ and U is maximal among all such possible allocations.

2.5 QoS Considerations in Video-Conferencing

We shall use a video-conferencing system named *RT-Phone* [7] presented in Figure 1 as an example to illustrate Q-RAM. We shall focus primarily on managing resource allocations for the audio stream on a single node. The end-to-end delay encountered by an audio stream as a function of the CPU processing rate and the audio sampling rate is plotted in Figure 2-a. The variable on the x-axis is the periodic interval at which buffered audio packets are obtained from the sound hardware, processed and transmitted over the network. Each plotted line corresponds to a different sampling rate. For shorter

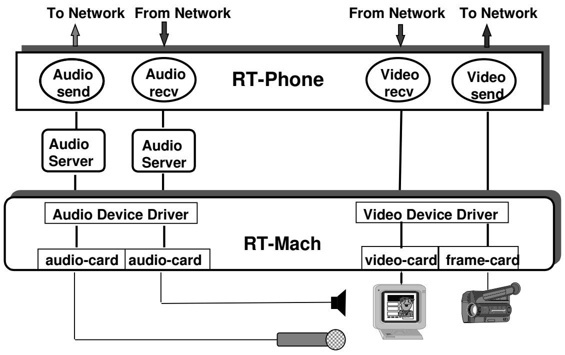


Figure 1: The Architecture of *RT-Phone*.

periods, the end-to-end delay is shorter and vice-versa. This plot also illustrates that with relatively little additional resources, the sampling rate can be increased and improved audio data quality can be obtained. The load imposed on the processor for the data points from Figure 2-a is plotted in Figure 2-b. The y-axis is now the CPU load; as can be seen, for shorter processing periods, the CPU load is high due to higher network packet processing costs (larger number of smaller packets) and higher context switching costs.

The QoS dimensions in this system (as described) are end-to-end delay representing timeliness and audio sampling rate representing data quality. The processing rate and the audio sampling rate can be changed independently of one another, and an increase in either leads to increases in utility of the video-conferencing system. Improvements in end-to-end delay from 200 *ms* to 50 *ms* generally tend to be perceived as much higher than improvements from 50 *ms* to 12.5 *ms*, i.e. the return on utility diminishes as more and more resources are added. The same applies to the sampling rate. The shapes of the utility functions[2] corresponding to these QoS dimensions are presented in Figure 2-c.

3 Resource Allocation in Q-RAM

In this section, we derive some basic properties of Q-RAM defined in the previous section.

We start with the simple case of making allocation decisions where there is only a single resource type and a single QoS dimension. We then extend this model to support multiple QoS dimensions. In each case, we state a property that needs to be satisfied for maximizing the total system utility, and/or present an algorithm which can find the optimal (or near-optimal) allocation.

3.1 A Single Resource and A Single QoS Dimension ($m = 1$ and $d = 1$)

Since there is a single resource and quality dimension, we can drop the subscripts associated with them. In this

[2] The utility assigned to an operating point for an application can be objective or subjective. In feedback control applications, control quality improves with sampling rates, and utility values can often be defined objectively. In multimedia applications, human perceptions saturate beyond a point, and utility values may be subjective. Relative utility values across applications would be based on system and application semantics, and will be, optionally, modifiable by the end-users.

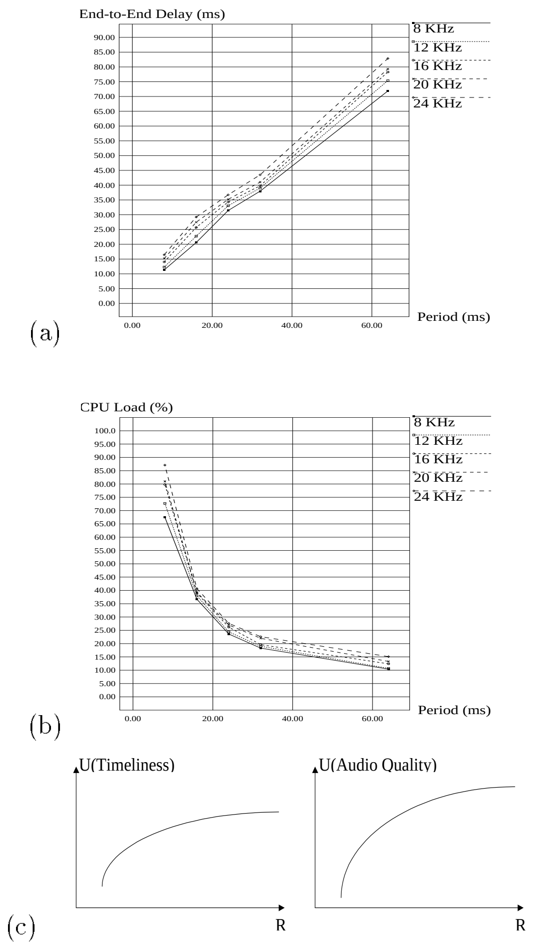


Figure 2: (a) End-to-end audio delay curves as a function of CPU processing rates and audio sampling rates. (b) The CPU load for audio processing as a function of CPU processing rates and audio sampling rates. (c) Audio utility functions for Timeliness and Data quality.

case, we have $U_i = U_i(R), 1 \leq i \leq n$, where R is the amount of resource allocated to τ_i. The minimum resource allocation needed to satisfy τ_i is R_i^{min}.

To illustrate our approach, we make the further assumption that the utility functions $U_i = U(R)$ are twice continuously differentiable and concave, that is $\frac{d^2U_i}{dR^2} = U_i'' \leq 0$ for $R > R_i^{min}$. By convention, we assume $U_i(R) = 0$ for $0 \leq R \leq R_i^{min}$.

It is very convenient to transform the resource allocation problem. Since we assume that all minimal application resource requests can be met, we can focus on the allocation of the excess resources available. Consequently, we can, without loss of generality, assume that $R_i^{min} = 0, \forall i = 1$ to n and reduce the quantity of available resources by that amount. In our subsequent analysis, we assume that this transformation has been made and require only that $R_i \geq 0$ and $\sum_{i=1}^{n} R_i = R$, where R is the remaining quantity of resources left to allocate.

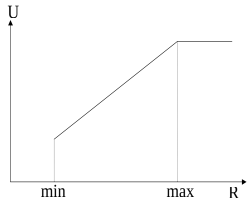


Figure 3: A linear utility function with min and max requirements.

The goal is to determine the values of $R_1, R_2, \cdots, R_n$ such that the total system utility, $\sum_{i=1}^{n} U_i(R_i)$, is maximized subject to the constraint $\sum_{i=1}^{n} R_i \leq R$. The following theorem provides a necessary condition for an allocation to be optimal.

Theorem 1 *A necessary condition for a resource allocation to be optimal is* $\forall i, 1 \leq i \leq n, R_i = 0$ *or for any* $\{i, j\}$ *with* $R_i > 0$ *and* $R_j > 0$, $U_i'(R_i) = U_j'(R_j)$.

Proof: The result is a standard conclusion of the Kuhn Tucker theorem (see [12], chapter 5). To understand the intuition behind the results, suppose that for some $i \neq j$, let $R_i > 0$, $R_j > 0$ and $U_i'(R_i) > U_j'(R_j)$.

Since $R_j > 0$, an infinitesimal amount of $\mathbf{R}$ can be subtracted from application τ_j and added to application τ_i. Since $U_i'(R_i) > U_j'(R_j)$, the total system utility will increase. This contradicts the assumption that the allocation was optimal.□

Remark: It should be noted that it is possible that all applications except one can receive zero resource allocations[3], and this one application consumes all the available resource quantity since the slope of its utility function is the highest.

Remark: If the utility functions were not smooth, then this result requires some modification. To see this, suppose U_1 consists of two line segments with slope s_1 on $[0, L]$, then s_2 on $[L, \infty)]$. Now suppose that U_2 is linear with slope s_3 with $s_1 > s_3 > s_2$, while all other utility functions have slopes which are less than s_2. If the amount of resources available exceeds L, then the optimal allocation will be to give L to the first application, all the rest to the second application and none to any others. This results in a situation of unequal slopes. If, on the other hand, the utility functions are smooth, this cannot happen.

3.1.1 A Special Case of Linear Utility Functions

As a special case, consider the utility curve of Figure 3. The utility curve is linear from R_i^{min} to a maximum resource requirement R_i^{max} beyond which it becomes flat. This utility curve is practical in the sense that many non-critical systems such as desktop multimedia applications can gain from its simplicity and resulting efficiencies. We refer to this special class of utility functions as *min-linear-max* functions. The following corollary provides a necessary condition for a resource allocation to be optimal for

[3] Recall that the resource allocations are normalized and that each application has already been allocated sufficient resources to satisfy its minimum requirement.

min-linear-max utility functions.

Corollary 1 *A necessary condition for a resource allocation to be optimal for min-linear-max utility functions is* $\forall i, 1 \leq i \leq n, R_i = 0$*, or* $R_i = R_i^{max}$*, or for any* $\{i,j\}$ *with* $0 < R_i < R_i^{max}$ *and* $0 < R_j < R_j^{max}$*,* $U_i'(R_i) = U_j'(R_j)$*.*

Proof: The corollary follows from the conditions of Theorem 1, and from the fact that no utility is gained by allocating even an infinitesimal resource to an application τ_i beyond R_i^{max}.□

Remark: It is possible that there exists only one application τ_i which has $0 < R_i < R_i^{max}$, i.e. i, j can refer to the same application in the statement of Corollary 1.

3.1.2 An Algorithm to Determine $\mathbf{U}_{max}$

An algorithm to determine the optimal resource allocation R_i for each application to obtain $\mathbf{U}_{max}$ is given below. We assume that each application has already been allocated its minimum resource requirement. By assumption A3, sufficient resources should be available for this allocation. Consequently, we determine the optimal additional allocation to each application, $R_i \geq 0, 1 \leq i \leq n$, subject to $\sum_{i=1}^{n} R_i \leq R$.

1. Let the current normalized allocation of the resource to τ_i be $R_i, 1 \leq i \leq n$. Let the unallocated quantity of the available resource be R^l. Compute $(U_1'(R_1), \cdots, U_n'(R_n))$.
2. Identify (a) the subcollection of applications with largest value of $U_i'(R_i)$, (b) the number of applications in that subcollection (denoted by k), and (c) the application (denoted by j) with the second largest value of this quantity if any such application exists. If the largest value of $U_i'(R_i)$ is 0, then stop. No further allocation will increase system utility and spare resources are available.
3. Increase R_i for each of the members of the subcollection so that their values of $U_i'(R_i)$ decrease but continue to be equal until either (i) this value becomes equal to the second largest value or (ii) the additional resources added to this subcollection equal R^l. In case (ii), stop as all resources have been optimally allocated.
4. In case (i), one or more new applications should be added to the subcollection. Return to step 1.

4 A Single Resource and Multiple QoS Dimensions ($m = 1$ and $d > 1$)

An application can have multiple QoS dimensions (i.e. $d > 1$). For example, the RT-Phone example has two QoS dimensions, audio data quality (which increases with audio sampling rate) and end-to-end delay (which decreases with increases in processing rate). The resource allocation for systems with multiple quality dimensions depends upon the nature of the relationship between the dimensions themselves. In this section, we classify the relationships between QoS dimensions, discuss their effects and study the resource allocation problem under various conditions. We provide optimal allocations when possible, and provide a greedy algorithm in another case.

4.1 Relationships between QoS Dimensions

The inter-relationship between QoS dimensions directly impacts the nature of the utility functions. We study two kinds of relationships among QoS dimensions:

Independent dimensions: Two QoS dimensions, Q_a and Q_b, are said to be independent of one another if a quality increase along Q_a (Q_b) does *not* increase the resource demands to achieve the quality level previously achieved along Q_b (Q_a). An example is using different compression schemes on an audio stream but each scheme generates the exact same amount of data. As a result, the processing resources needed to encrypt the data remain the same. If the encryption scheme is changed to consume more resources, the audio compression demands would remain the same. Therefore, security and audio data quality can be considered to be independent QoS dimensions in this system.

Dependent dimensions: A QoS dimension, Q_a, is said to be dependent on another dimension, Q_b, if a change along the dimension Q_b will increase the resource demands to achieve the quality level previously achieved along Q_a. In the RT-Phone system, if the audio sampling rate is increased, the data volume increases and the CPU time needed to process the data increases[4].

Remark: Two QoS dimensions Q_a and Q_b can both be dependent on a third dimension Q_c. For example, if video quality is improved by increasing the size of the image, both processing capacity and network bandwidth demands would increase. As a result, both timeliness and packet loss QoS dimensions would be affected.

4.2 Dealing with Independent QoS Dimensions

Suppose that the d QoS dimensions are independent of one another. In this case, each QoS dimension offers its own utility to the system and can be varied independent of the other dimensions. In this case, the dimensional utilities of the applications are additive. That is, $U_i = \sum_{k=1}^{d} U_{i,k}$. The resource allocation problem then is equivalent to the single QoS dimension problem of Section 3.1 with $n * d$ applications $\{\tau_1', \tau_2', \cdots, \tau_{n*d}'\}$, where $\{\tau_1', \tau_2', \cdots, \tau_d'\}$ correspond to the d dimensions of τ_1, $\{\tau_{d+1}', \tau_{d+2}', \cdots, \tau_{2d}'\}$ correspond to the d dimensions of τ_2 and so on. The optimal resource allocations can now be determined using the algorithm described in Section 3.1.2.

4.3 Dependent QoS Dimensions with Continuous Values

Suppose that one or more QoS dimensions are independent, but the quality on each dimension can be any value within an interval. We now illustrate the general approach using the special case $d = 2$, but the approach

[4]This increase in CPU load is not linear, however, as can be inferred from Figure 2-b.

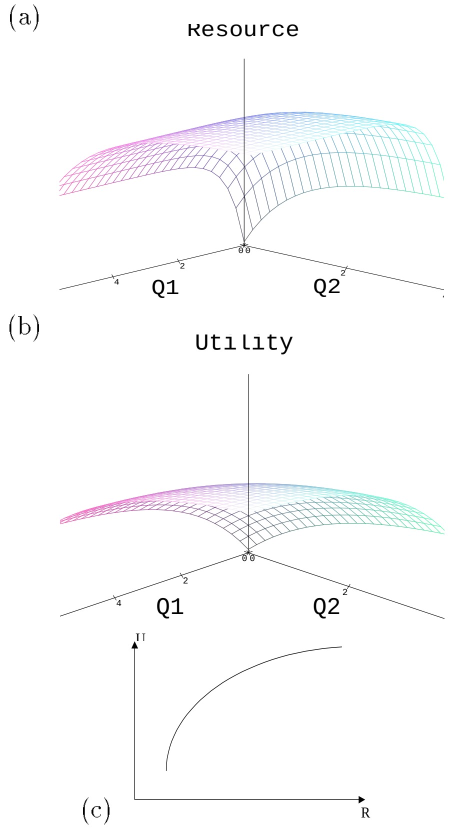


Figure 4: (a) The Resource Consumption Surface for Two QoS dimensions. (b) The Utility Surface as a function of two Qos Dimensions. (c) The final (univariate) utility function for two QoS dimensions and a single resource.

remains the same for $d > 2$. The resource demand for every point p along the QoS dimension Q_1 and every point q along the QoS dimension Q_2 is plotted first. This defines a resource consumption surface along Q_1 and Q_2, an example of which is provided in Figure 4-a. The utility to the system for any pair $\{p, q\}$ of points along QoS dimensions Q_1 and Q_2 respectively is plotted next. This yields a utility surface, an example of which is illustrated in Figure 4-b. The contours of $R = k$ from the resource consumption surface are then projected to the utility surface. The maximum utility values for each $R = k$ contour projection finally yield a single (maximal) utility function as a function of R. For example, the utility function from the surfaces of Figures 4-a and 4-b yield the shape shown in Figure 4-c. The net result is that the multi-dimensional resource allocation problem gets reduced to the single-QoS dimension problem.

If the resulting univariate utility function is twice continuously differentiable and concave (or min-linear-max), the algorithm of Section 3.1.2 can be applied to obtain the optimal resource allocation.

4.4 Dependent QoS Dimensions With Discrete Options

We now consider special cases, where one of the QoS dimensions is not only dependent on another (base) QoS

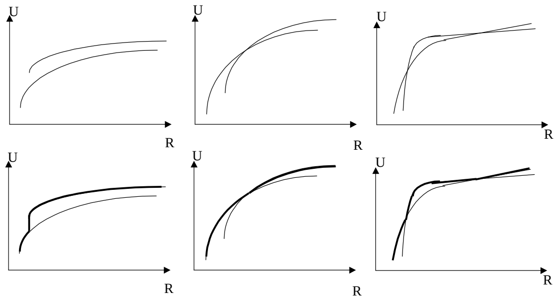


Figure 5: Three sets of U_r, U_e and an aggregate U_a for a dependent binary QoS Dimension.

dimension, but is also discrete in nature. For the sake of illustration, we shall assume that $d = 2$, yielding one independent (base) QoS dimension and one QoS dimension dependent on the former. We shall first consider the binary case where the quality of the dependent dimension is either available or not available. We then consider the case where the quality along the dependent dimension can be any one of multiple discrete values.

4.4.1 Using Dependent Binary QoS Dimensions

Consider again the application sampling microphone input and transmitting an audio stream. In the following, we use the suffixes a, r and e to represent audio, raw audio and encrypted audio respectively. Increasing the audio sampling rate increases the audio quality and the amount of data to be processed. Let R_r be the CPU resource allocated to the processing of this raw data. We have $R_r = g(SamplingRate)$. Assuming that $g(\cdot)$ is monotonic, $U_r = f(R_r)$ has the same shape as Figure 4-c.

Suppose that the audio data will also be encrypted. Now, additional processing per block of audio data (and correspondingly the CPU resource) will be needed. This additional resource consumption scales linearly with the sampling rate. It is reasonable to assume that a constant utility gain Δ is added to the system with encryption. However, since R_r is needed for processing the audio data without encryption, a *larger* value R_e would need to be allocated for encrypting and processing the same amount of audio data. We therefore have $R_e = \Gamma_e * R_r$, where Γ_e is a constant > 1.0.

The utility function U_e therefore has the form $f(\frac{R_e}{\Gamma_e}) + \Delta$. Therefore, the origin of U_e is offset both vertically and horizontally from U_r. The vertical offset is Δ, and the horizontal offset is $(\Gamma_e - 1) * R_r^{min}$. Since $\Gamma_e > 1$, the slope of U_e is always smaller than that of U_r. If U_r is continuous and concave, U_e is also continuous and concave.

The aggregate utility function for the audio application is given by $U_a = \max(U_r, U_e)$. Three examples are provided in Figure 5; U_r is the thin line starting lower and more to the left, U_e is the other thin line, and U_a is the bold line. It must be noted that encryption is not possible for resource allocations less than $(R_r^{min} * \Gamma_e)$. For larger allocations, encryption is possible but it may or may not yield higher utility. For example, the aggregated application utility function of Figure 5-c yields a higher utility without encryption initially, then with encryption, and then without encryption again. Also, in the general case, U_a will only be piecewise continuous and concave.

4.4.2 Using Dependent 'n-ary' QoS Dimensions

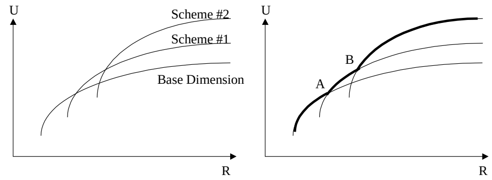


Figure 6: Example of using an n-ary QoS Dimension.

In Section 4.4.1, we assume that a second QoS dimension (namely encryption) is applied as a binary function: it is either available or not available. Such a binary scheme is applicable when only a single scheme (such as encrypting always using a 48-bit public key) is used. However, suppose that more than one scheme is available along this QoS dimension (such as encrypting using a 64-bit key, a 128-bits key, etc. or using a different cryptographic scheme). For convenience, we introduce the following notation.

Notation: Let the QoS dimension Q_k be dependent on another and have $s+1$ discrete schemes. By convention, we adopt scheme 0 to represent the absence of the dimension. The utility gain constant provided by supporting scheme p, $0 \leq p \leq s$ is denoted by $\Delta_{k,p}$. The *resource scaling factor* (with respect to the base dimension) for supporting scheme p, $0 \leq p \leq s$ is denoted by $\Gamma_{k,p} \geq 1.0$. The *overhead factor* of scheme p, $0 \leq p \leq s$ is denoted by $\gamma_{k,p} = (1.0 - \Gamma_{k,p}) \geq 0.0$. By convention, we have $\Delta_{k,0} = 0$, $\Gamma_{k,0} = 1.0$, $\gamma_{k,0} = 0.0$.

Each scheme p, $0 \leq p \leq s$, provides a correspondingly different increase in utility, $\Delta_{k,p}$, while also consuming a different amount of the CPU resource with a different $\Gamma_{k,p}$. The result is a family of utility curves, and the aggregated *application utility function* is the maximum of these curves. An example family of utility functions for a 3-ary dependent dimension and the resulting aggregated utility function are illustrated in Figure 6.

4.4.3 Linear Dimensional Utility Functions in the Dependent 'n-ary' Case

When min-linear-max dimensional utility functions are used for the independent QoS dimensions, the aggregated application utility function is piecewise linear when one of the QoS dimensions is 'n'-ary in nature. A sample set of the individual dimensional utility functions for two QoS dimensions, one independent and another dependent, and their aggregated application utility function are illustrated in Figure 7. Let the independent (base) dimension be Q_1 and the dependent dimension be Q_2. We have $R_{min} = 0.1$, $R_{max} = 0.2$ for Q_1 (same as scheme 0 for

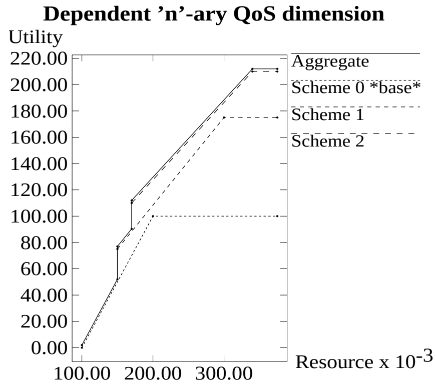


Figure 7: One 3-ary QoS dimension w/ min-linear-max.

Q_2). The corresponding utilities are given as 0 and 100 respectively. For Scheme 1 of the dependent dimension Q_2, we assume $\Delta_{2,1} = 75$, $\Gamma_1 = 1.5$ yielding $\gamma_1 = 0.5$. Correspondingly, $R_{min} = 0.15$, $R_{max} = 0.3$, and the utilities are given by (0+75=75) and (100+75=175). For Scheme 2, assume $\Delta_{2,2} = 110$, $\Gamma_{2,2} = 1.7$, $\gamma_{2,2} = 0.7$. We therefore have $R_{min} = 0.17$, $R_{max} = 0.34$, $U_{min} = 0 + 110 = 110$ and $U_{max} = 100 + 110 = 210$. The aggregate utility function for the application is the maximum of the previous three functions.

It is useful to note that this piecewise-linear application utility function has 3 kinds of discontinuities: *intersecting discontinuities* where dimensional utility lines intersect (point A in Figure 6), *vertical discontinuities* where the maximal dimensional utility line at a point starts above the other lines (at $R = 0.15$ and 0.17 in Figure 7), and *saturation discontinuities* where the maximum QoS point for a dimensional utility line beyond which the utility does not increase (at $R = 0.34$ in Figure 7). We now present a greedy algorithm which determines a near-optimal resource allocation under these conditions.

A greedy algorithm to obtain a good resource allocation R_i for each application in a system with all linear dimensional utility functions is as follows:

1. Assign to each application τ_i its minimum resource requirement R_i^{min}. By assumption A3, sufficient resources should be available for this allocation.
2. Normalize the utility function of each application (by left-shifting and down-shifting the utility curve such that it starts at the origin). Let the total quantity of available resource remaining be R.
3. Let the current normalized allocation of the resource to τ_i be R_i. Let the unallocated quantity of the available resource be R^l.
4. For *each* application $\tau_i, 1 \leq i \leq n$, compute the slopes on the application utility curve at R_i, *and* between the current allocation R_i and any and every discontinuity[5] in the region $R_i < R \leq R^l$. Let

[5]When there is a vertical discontinuity, pick the higher point.

this set of slopes for τ_i be represented by $\{s_i\}$. The size of this set of slopes is at least one.

5. Let the index of the application with the highest value of the element in $\{s_i\}$, $1 \le i \le n$, be p. If there are two or more such applications, pick one at random. Let this largest slope element of τ_p be s_p^{max}. Let the *additional* resource amount that needs to be allocated to τ_p to reach the discontinuity point corresponding to s_p^{max} be r.
6. If $s_p^{max} = 0$, stop. The unallocated resources will *not* increase system utility any further.
7. Allocate an additional (r) to τ_p increasing R_p by that amount. Reduce R^l by this same amount.
8. If $R^l = 0$, stop. Else, go to step 4.

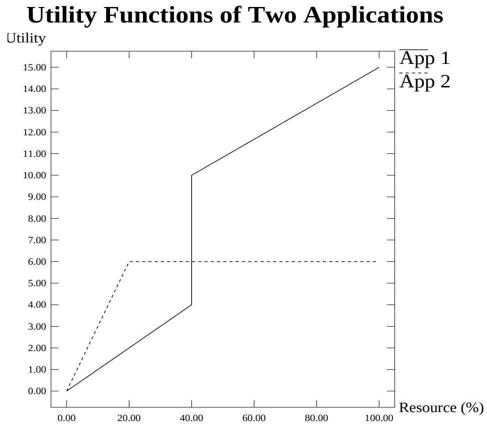
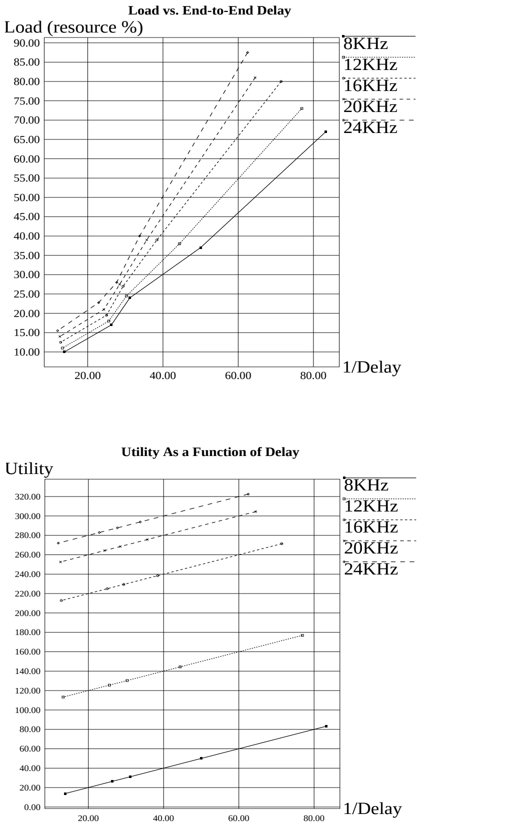


Figure 8: A counter-example of non-optimality of the greedy algorithm of Section 4.4.3).

<u>Remark</u>: We note that the above algorithm, while expected to do well in practice, does *not* always lead to an optimal resource allocation[6]. We now provide a counter-example illustrated in Figure 8 to show that this in indeed the case. Suppose that there are only two applications τ_1 and τ_2. Let the total amount of resource to be allocated be 0.5. $U_1(0)$ has a linear slope of 10, but a vertical discontinuity occurs at $R_1 = 0.4$ with $U_1(0.4) = 10$. The slope at $U_1(0.4)$ continues to be 10. But, the slope of $U_1(0)$ from 0 to the higher point of the vertical discontinuity, which is at 10, is 25. $U_2(0)$ has a linear slope of 30 and U_2 has a saturation discontinuity at $R_2 = 0.2$ (with $U_2(0.2) = 6$). The above greedy algorithm will first allocate 0.2 units of the resource to τ_2 (since its slope is the highest at a value of 30). The remaining 0.3 units will then be allocated to τ_1 yielding $U_1(0.3) = 3$. The total system utility achieved is therefore (6+3)=9. However, an optimal algorithm would allocate $R_1 = 0.4$ and $R_2 = 0.1$ yielding a total system utility of (10+3) = 13, which is higher than the utility achieved by the greedy algorithm.

[6] We are currently developing an algorithm that will find an optimal allocation and replace this greedy (sub-optimal) algorithm.

(a)



(b)

Figure 9: (a) The resource consumption function for RT-Phone. (b) Utility as a function of Timeliness (with a linear model of the value of timeliness).

4.5 Using Q-RAM in the RT-Phone Example

In this section, we apply Q-RAM to the RT-Phone system for the sake of illustration and also show how the real-time constraints can be satisfied. We first generate the resource consumption surface for the QoS dimensions end-to-end delay (represented as 1/delay) and audio quality (represented as sampling rate). From Figures 2-a and 2-b, we obtain the surface in Figure 9.a. We now assume that the utility of the timeliness QoS dimension is given by (1/delay). Let us now suppose that the audio quality dimension offers a constant utility *gain* at each sampling rate.

Using $U_g(\cdot)$ to represent the utility gain from a particular sampling rate, let us assume that $U_g(\text{8KHz}) = 0$, $U_g(\text{12KHz}) = 100$, $U_g(\text{16KHz}) = 200$, $U_g(\text{20KHz}) = 240$, $U_g(\text{24KHz}) = 260$ (yielding a tapering-off effect). The total utility at a given sampling rate is given by $U_{delay} + U_g$. The variation of total utility with end-to-end delay is plotted in Figure 9.b. From the curves of Figure 9, we obtain the (univariate) utility function of Figure 10.

4.5.1 Resource Allocation and Schedulability

Suppose the CPU resource has to be allocated among 10 applications with utility curves similar to those of Figure 10. First, all 10 applications will be allocated their min-

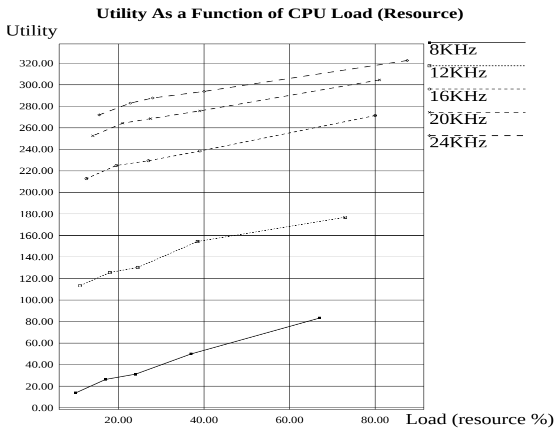


Figure 10: The RT-Phone utility function recommends the use of a 24KHz sampling rate for audio.

imum resource requirements. Next, additional resource allocations will be made only to the application with the highest utility slope. If any CPU cycles remain after that application reaches its maximum requirement, only then would they be allocated to the application with the next higher slope. This is repeated until no more CPU cycles are available. Let the final CPU allocation to application i be R_i. Its corresponding CPU load and processing rate from Figures 2-a and 2-b yield the (computation time, period) pair. These pairs can then be scheduled using the earliest deadline scheduling algorithm.

5 Concluding Remarks

We have presented a QoS-based Resource Allocation Model (Q-RAM) that allows the utility derived from a system to be maximized by making resource allocations such that the different needs of concurrently running applications are satisfied. Each application has a minimal resource requirement, but can adapt its behavior if given more resources and provide additional utility. Each application also needs to satisfy QoS metrics along multiple dimensions such as timeliness, cryptographic security, reliable packet delivery and data quality. Finally, each application may need to obtain access to multiple resource types in order to meet its QoS constraints. We have provided optimal (or near-optimal) resource allocation schemes for applications which need a single resource, but need to satisfy one or more QoS dimensions. A video-conferencing system with timeliness, audio quality and encryption constraints is used as an example to motivate and apply Q-RAM.

We are pursuing several avenues as future work. First, optimal schemes are needed for applications with multiple QoS dimensions (see Section 4.4.3) and for allocation of multiple resources. Second, the underlying OS/kernel for Q-RAM must not only support flexible resource management schemes but also provide feedback to the Q-RAM manager about available resources and resource consumption by various application threads. The run-time overhead for these actions are also yet to be studied in detail. Finally, Q-RAM is based on single-node systems and needs to be extended to distributed systems.

Acknowledgments

The authors would like to thank other Amaranth project members at Carnegie Mellon University, including Carol Hoover, Pradeep Khosla, Phil Koopman and Lui Sha. The Amaranth project is defining a comprehensive framework for QoS management along multiple quality dimensions, and its goals include the construction of system prototypes and applications. The authors would also like to thank John Wilkes of Hewlett Packard and Tom Lawrence of Rome Air Force Laboratories for insightful discussions on QoS-based resource allocation.

References

[1] T. Baker. Stack-based scheduling of realtime processes. *Journal of Real-Time Systems*, 3(1):67–100, March 1991.

[2] K. G. Shin D. D. Kandlur and D. Ferrari. Real-time communication in multi-hop networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 1044–1056, Oct 1994.

[3] R. Guérin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas in Communications*, September 1991.

[4] K. Jeffay. Scheduling sporadic tasks with shared resources in hard real-time systems. Technical report, TR90-038, Department of Computer Science, University of North Carolina at Chapel Hill, November 1989.

[5] M. B. Jones and P. J. Leach. Modular real-time resource management in the rialto operating system. Technical Report MSR-TR-95-16, Microsoft Research, Advanced Technology Division, May 1995.

[6] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate-Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, 1993. ISBN 0-7923-9361-9.

[7] C. Lee, R. Rajkumar, and C. Mercer. Experiences with processor reservation and dynamic qos in real-time mach. *In the proceedings of Multimedia Japan 96*, April 1996.

[8] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhancing aperiodic responsiveness in a hard real-time environment. *IEEE Real-Time System Symposium*, 1987.

[9] C. L. Liu and Layland J. W. Scheduling algorithms for multiprogramming in a hard real time environment. *JACM*, 20 (1):46 – 61, 1973.

[10] J. W. S. Liu, K-J Lin, R. Bettati, D. Hull, and A. Yu. *Use of Imprecise Computation to Enhance Dependability of Real-Time Systems.* Kluwer Academic Publishers, 1994.

[11] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[12] A. L. Peressini, R. E. Sullivan, and Jr. J. J. Uhl. *Convex Programming and the Karish-Kuhn-Tucker conditions*, chapter 5. Springer-Verlag, 1980.

[13] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach.* Kluwer Academic Publishers, 1991. ISBN 0-7923-9211-6.

[14] D. Seto, J. P. Lehoczky, L. Sha, and K.G. Shin. On task schedulability in real-time control systems. *IEEE Real-Time System Symposium*, December 1996.

[15] J. A. Stankovic and K. Ramamritham. The design of the spring kernel. In *Proceedings of the Real-Time Systems Symposium*, Dec 1987.

[16] E. M. Atkins T. F. Abdelzaher and Kang Shin. Qos negotiation in real-time systems and its application to automated flight control. In *The Proceedings of the IEEE Real-time Technology and Applications Symposium*, June 1997.

[17] W. Zhao, K. Ramamritham, and J. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, Aug. 1987.

[18] T. F. Lawrence. The Quality of Service Model and High Assurance. *Workshop on High Assurance Systems*, July 1997.