On Quality of Service Optimization with Discrete QoS Options*

Chen Lee, John Lehoczky, Ragunathan (Raj) Rajkumar, Dan Siewiorek
Carnegie Mellon University
{clee, raj+, dps}@cs.cmu.edu, jpl@stat.cmu.edu

Abstract

We present a QoS management framework that enables us to quantitatively measure QoS, and to analytically plan and allocate resources. In this model, end users' quality preferences are considered when system resources are apportioned across multiple applications such that the net utility that accrues to the end-users is maximized. In [23][24], we primarily worked with continuous QoS dimensions, and assumed that the utility gained by improvements along a QoS dimension were always representable by concave functions. In this paper, we relax both assumptions. One, we support discrete QoS operating points. Two, we make no assumptions about the concavity of the utility functions. Using these as the basis, we tackle the problem of maximizing system utility by allocating a single finite resource to satisfy the QoS requirements of multiple applications along multiple QoS dimensions. We present two near-optimal algorithms to solve this problem. The first yields an allocation within a known bounded distance from the optimal solution, and the second yields an allocation whose distance from the optimal solution can be explicitly controlled by the QoS manager. We compare the run-times of these near-optimal algorithms and their solution quality relative to the optimal allocation, which in turn is computed using dynamic programming. These detailed evaluations provide practical insight into which of these algorithms can be used online in real-time systems.

1. Introduction

Quality of Service (QoS) control is receiving widespread attention in commercial markets as well as computer network and real-time multimedia system research. Typically, service characteristics in existing multimedia and networked systems are fixed when systems are built, therefore they often do not give users any real influence over the QoS they can obtain. On the other hand, multimedia applications and their users can differ enormously in their requirements for service quality and the resources available to them at the time of application use. Therefore, there is an increasing need for customizable services that can be tailored for the end users' specific requirements.

In the meantime, new and improved systems such as the one proposed by the Amaranth project at Carnegie Mellon University [1] are placing more and more complex demands on the quality of service that are reflected in multiple criteria over multiple quality dimensions. These QoS requirements can be objective in some aspects and subjective in others. Moreover, because of the manifold and subjective nature of user quality demands, it is very hard to measure whether the provided quality fulfills the stated demands without guidance and input from end clients.

One issue is *QoS Tradeoffs* where a user of an application might want to emphasize certain aspects of quality, but not necessarily others. Users might tolerate different levels of service, or could be satisfied with different quality combination choices, but the available system resources might only be able to accommodate some choices but not others. In situations where a user is able to identify a number of desirable qualities and rate them, the system should be able to reconcile these different demands to maximize the user's preference and to make the most effective use of the system. So it is important for a system to provide a large variety of service qualities and to accommodate specific user quality requirements and delivery as good service as it can from the users' perspective.

An issue related to QoS Tradeoff is *Resource Tradeoff*. In this case, the tradeoff refers to reconciling or balancing competing resource demands. Resource Tradeoff is often transparent to the user but can be of great help in accommodating user requirements including QoS Tradeoff, especially when the availability of several different resources is not balanced. It arises when an application is able to use an excess of one resource, say CPU power, to lower its demands on another, say network bandwidth, while maintaining the same level of QoS. For example, video conferencing systems often use compression schemes that are effective, but computationally intensive, to trade CPU time for network bandwidth. If the bandwidth is congested on some intermediate links (which is often the case), this ben-

*This research was supported in part by the DARPA under agreements E30602-97-2-0287 and N66001-97-C-8527.

efits the system as a whole. In the case of a mobile client with limited CPU and memory capacity but sufficient link speed with a nearby intermediate powerful server, computationally expensive speech recognition, silence detection and cancellation, and video compression could be carried out on the nearby server. For proxy servers which act as transcoders/transceivers besides caching data, the proxy servers can distill data for low bandwidth clients (when both server and client have fast CPU, memory and disk bandwidth, but the network link speed in between is limited).

1.1. Related Work

Research on Quality of Service for multimedia applications has gained significant momentum over the last few years. Much research has been being conducted on the end-system or end-to-end architectures for QoS support [11, 5, 16, 14], and much more work has been conducted on link, network and transport layer ([28], for instance).

Most of this research has been focused on low-level system mechanisms. The authors consider parameters such as period, buffer size, jitter, bandwidth and so on. No doubt these are important issues and factors for QoS control, but they are hardly tangible for the ultimate end-users who experience the resulting QoS.

Research on adaptive QoS control (e.g. [20]) brings us a step closer to the QoS support from a user's perspective by providing a mechanism in an application to accommodate potential dynamic changes in the operating environment. But these mechanisms are still mainly system-oriented in that a user has limited influence over the quality of the service to be delivered or adapted.

In coping with the shortage of QoS support from an end-user point of view, we proposed a basic framework [15, 17, 23] that enables the end users to give guidance on the qualities they care about and the tradeoffs they are willing to make under potential resource constraints. Working from the user's perspective and maximizing the user perceived quality or utility has also been addressed in [13, 3, 4]. In [13], a user-centric approach is taken, where a user's preferences are considered for application runtime behavior control and resource allocation planning. Example preferences include statements that a video-phone call should pause a movie unless it's being recorded and that video should be degraded before audio when all desired resources are not available. These are useful hints for high-level QoS control and resource planning, but are inadequate for quantitatively measuring QoS, or analytically planning and allocating resources.

The notion of using utility functions to represent varying satisfaction with QoS changes is certainly not new. Jensen et al. [12] and Locke [18] are perhaps among the first to study "value functions" to represent the benefit of different completion times of a task. Their value function model is a utility function along the latency quality dimension of real-time tasks. Our model can be viewed as extending this notion to include quality dimensions other than timeliness. Another utility model for QoS control is used in [3]. The authors propose a mechanism for QoS (re-)negotiation as a way to ensure graceful degradation. The authors suggest that a user should be able to express, in his/her service requests, the spectrum of QoS levels the user can accept from the provider, as well as the perceived utility of receiving service at each of these levels. But the authors did not address the resource tradeoff problem. Also, no specification method and mechanism is provided to facilitate utility data acquisition. Interesting research is being conducted in [4], where the authors present a framework for the construction of network-aware applications. The basic idea is to allow an application to adapt to its network environment, e.g. by trading off the volume (and with it the quality) of the data to be transfered and the time needed for the transfer. Their mechanism coincides with one of our schemes for implementing the resource tradeoff ($r \models_i q$). The model defined in [17] can be considered a generalization of [4].

2. Problem Taxonomy and Modeling

2.1. Quality Dimensions

Consider a video-conferencing system which deals with real-time audio and video data streams being encrypted and transmitted across potentially unreliable networks. In this context, we consider the following example quality dimensions with their corresponding dimensional space ordered from worst to best:

- Cryptographic Security (encryption key-length)
  - 0(off), 56, 64, 128
- Data Delivery Reliability, which could be
  - maximum packet loss : in percentage
  - expected packet loss : in percentage
  - packet loss occurrence : in probability
- Video Related Quality
  - picture format[1]: SQCIF, QCIF, CIF, 4CIF, 16CIF
  - color depth(bits): 1, 3, 8, 16, 24, ...
    black/white, grey scale to high color
  - video timeliness — frame rate(fps): 1, 2, ..., 30
    low rate animation to high motion picture video
- Audio Related Quality
  - sampling rate(kHz): 8, 16, 24, 44, ...
    AM, FM, CD quality to higher fidelity audio
  - sample bit(bits): 8, 16, ...

[1] The choices listed here come from [10] [27]. Other standards, such as MPEG could have been used instead.

– audio timeliness, or end-to-end delay(ms)
…, 100, 75, 50, 25, …

The specification above contains ellipses "…" to indicate that more choices could have been listed. Ignoring extra choices for a moment, the total number of different choices (quality points) in this example (a single option in data delivery reliability, encryption on or off, and 30 frame rates could be chosen) will be

$$2 \times 1 \times 5 \times 5 \times 30 \times 4 \times 2 \times 4 = 48000$$

With this many quality points it would be completely out of the question to have the user specify the quality on a point-by-point basis. Therefore a pragmatic scheme is needed to address the issue.

2.2. Problem Taxonomy

We assume that multiple applications similar to the one described in the previous subsection can co-exist in a system. Bearing in mind that discrete choices are available along each QoS dimension, we classify our problem based on resources and QoS dimensions as follows:

- Single Resource Single QoS Dimension: SRSD
- Single Resource Multiple QoS Dimension: SRMD
- Multiple Resource Single QoS Dimension: MRSD
- Multiple Resource Multiple QoS Dimension: MRMD

Since SRMD is a superset of SRSD, and MRMD a superset of MRSD, SRMD and MRMD can be treated directly. In this paper, we focus only on SRMD. The reason for addressing SRMD is that we could develop efficient schemes that might not be easily achievable for MRMD. The schemes we have for SRMD readily lead us to a QoS-driven single resource allocation when only a single resource is of concern (either it is the only resource under consideration, or it is relatively more scarce and other resources are abundant). For instance, these schemes can be used for QoS-driven disk, memory, network bandwidth as well as for processor scheduling.

2.3. Problem Formulation

Consider a system with multiple independent applications and multiple resources. Each application, with its own quality-of-service requirements, contends with others for finite system resources. Let the following be given

$T_1, T_2, \ldots, T_n$	— tasks (or applications)
$R_1, R_2, \ldots, R_m$	— shared system resources
$Q_{i1}, Q_{i2}, \ldots, Q_{id_i}$	— QoS dimensions for task T_i

Each R_i is a set of non-negative values representing the possible allocation choices of the ith shared resource. The set of possible resource vectors, denoted as R, is given by $R = R_1 \times \cdots \times R_m$. Each shared resource is finite, so we also have $r^{\max} = (r_1^{\max}, \ldots, r_m^{\max})$.

Similarly, each Q_{ij} is a finite set of quality choices for the ith task's jth QoS dimension, and we define the set of possible quality vectors by $Q_i = Q_{i1} \times \cdots \times Q_{id_i}$.

Associated with each T_i is an *task profile*, which consists of an *application profile* and a *user profile*. An Application profile comes from an application designer, while a user profile provides user-specific quality requirement associated with each session. For simplicity, we will not distinguish the two sources and use task profile from now on.

A task profile for T_i consists of:

- Quality Space — Q_i
- Quality Index — a bijective function
  $$f_{ij} : Q_{ij} \to \{1, 2, \ldots, |Q_{ij}|\}$$
  that preserves the ordering, i.e., if q_1 is "better than" q_2, then $f_{ij}(q_1) > f_{ij}(q_2)$.
- Dimension-wise Quality Utility — $u_{ij} : Q_{ij} \to \mathbb{R}$
- *Application Utilities* — a rate of service measure
  $$u_i : Q_i \to \mathbb{R}$$
  It could be defined as a weighted sum of u_{ij}
  $$u_i(q_i) = \sum_{j=1}^{d_i} w_{ij} u_{ij}(q_{ij})$$
  We require that u_i is non-decreasing and non-negative.
- *Resource Profile*: a relation between R and Q_i
  $$r \models_i q$$
  which describes a list of potential resource allocation schemes to achieve each quality point q.

Note that both R and Q_i have partial orderings which $\models_i$ must respect. That is, if $r_1 \models_i q_1$, $r_2 \models_i q_2$, and $r_1 > r_2$, then we will have $q_1 \not< q_2$. This partial ordering is required to ensure that utility is non-decreasing with respect to resources. In other words, more resources should not lead to reduced quality (and thus utility), which is reasonable and natural.

It is important to note that we can only define a relation but not a function between Q_i and R. For a given value of q, multiple resource allocation schemes could be used to achieve the same level of quality; likewise, for a given resource allocation, one could use the resource(s) to improve different QoS dimensions, which could yield different quality results. Furthermore, a user could specify its

QoS Constraint

which is the minimum QoS requirement specification

$$q_i^{\min} = (q_{i1}^{\min}, q_{i2}^{\min}, \ldots, q_{id_i}^{\min}).$$

When the minimum requirements cannot be satisfied, the user of task T_i might choose not to run T_i at all. Alternatively we could let the user implicitly specify the $q_i^{\min}$ through utility functions by setting $u_i(q) = 0$ for all $q < q_i^{\min}$. We have yet to complete a user-interface study to decide whether this approach will compromise the simplicity of the user-interface. For now, we will use this QoS Constraint approach.

For the overall system, with multiple applications possibly requiring multiple resources, we have the

System Utility
$u : Q_1 \times \cdots \times Q_n \rightarrow \mathbb{R}$, which could be defined as:

- A (weighted) sum of *Application Utilities*
$$u(q_1, \ldots, q_n) = \sum_{i=1}^{n} w_i u_i(q_i)$$
for *differential services,* where u_i is non-decreasing, and $0 \leq w_i \leq 1$ could be the priority[2] of T_i, or
- $u = u^*$, where
$$u^*(q_1, \ldots, q_n) = \min_{i=1\ldots n} u_i(q_i)$$
for *"fair" sharing.*

The goal is to assign qualities (q_i) and allocate resources (r_i) to tasks or applications such that the system utility u is maximized. Therefore we have the following *Problem Function* formulation

$$\begin{aligned}
\text{maximize} \quad & u(q_1, \ldots, q_n) \\
\text{subject to} \quad & q_i \geq q_i^{\min} \text{ or } q_i = 0\,, \quad i = 1, \ldots, n, \\
& \qquad \textbf{(QoS Constraints)} \\
& \sum_{i=1}^{n} r_{ij} \leq r_j^{\max}\,, \qquad j = 1, \ldots, m, \\
& \qquad \textbf{(Resource Constraints)} \\
& r_i \models_i q_i\,, \qquad i = 1, \ldots, n. \\
& \qquad \textbf{(Resource Profiles)}
\end{aligned}$$

3. User Specification Interface

At the core of our QoS optimization system lies the QoS specification. First, it is important that we provide powerful and semantically rich QoS specifications that the system and the user can use for service optimization. Equally important we need to provide a *user-friendly interface* that facilitates specification acquisition.

The reason for the emphasis on QoS specification and interface design might not be obvious, but the reader should see the point shortly as the quality dimensions of typical multimedia systems, QoS tradeoff and resource tradeoff issues are presented.

[2]Note that the algorithms or schemes presented in this paper are for the weighted sum where the weights are set to 1 for simplification to present the algorithms.

3.1. Application Utility and QoS Tradeoff

Application utility functions are conceptually easy to imagine but difficult to construct. As pointed out in Section 2.1, it is clearly infeasible to make the user specify the utility of every quality choice on a point-by-point basis. There are simply too many choices. Instead, one could make the user specify the utility of selected points and then interpolate in order to obtain the utility of the rest. This might work well in the one-quality dimension case, but in the multi-dimensional case one would need a dense set of selected points and therefore again need too many points.

While we would like a user to provide the service optimization system with the semantically rich service requirement specification so that the optimization module can best accommodate the user's request, we also want to ensure that methods and mechanisms are in place in the system that will facilitate the delivery of these specifications from the user. In other words, we want to develop a measure and merit scheme as well as a reasonably user-friendly interface that will pose less of a burden on the user without sacrificing the semantically rich capability of the specification interface. We therefore propose a QoS index model from which the dimension-wise quality utility functions are defined.

3.2. Quality Index

Certain quality dimensions, such as frame rate, have easily defined utility functions while others, such as picture format, audio sampling rate and end-to-end delay, are in non-numeric, non-uniform, or non-increasing order which require a quality to numeric mapping. Therefore the Quality Index is introduced, which maps qualities to indices in order of increasing quality.

Let us illustrate the concept of the Quality Index through an example. Considering a task T_i with seven QoS dimensions. Due to limited space, let us look at only three selected QoS dimensions:

Picture format: Assume the H263 [10] standard format

Format:	SQCIF	QCIF	CIF	4CIF	16CIF
Quality Index:	1	2	3	4	5

Therefore Quality Index $Q_{i1} = \{1, 2, 3, 4, 5\}$.

Audio sampling rate: Assume audio sampling rates range from AM to CD quality.

Sampling rate (kHz):	8	16	24	44
Quality Index:	1	2	3	4

Thus we have $Q_{i5} = \{1, 2, 3, 4\}$.

End-to-end delay: Assume that end-to-end delays range from 125 ms to 25 ms in steps of 25 ms.

Delay (ms):	125	100	…	25
Quality Index:	1	2	…	5

Since high numbers for end-to-end delay are worse than low ones, $Q_{i7} = \{1, 2, \ldots, 5\}$ maps high numbers to low indices.

3.3. Dimension-wise Utilities

When many quality dimensions are involved, it is often very difficult for a user to express quality preferences. Quality points in multi-dimensional cases do not have a complete ordering. The individual dimensions, however, do. Moreover, some common properties associated with dimensional quality utility are observed including: non-decreasing, often quasi-continuous and piecewise concave. Figure 1 depicts some typical utility function shapes. We therefore provide the user with the capability to specify dimension-wise quality utilities. As a result, the application utility can then be defined as a weighted sum of dimension-wise utilities. This creates an interesting issue regarding how weights are assigned. We currently use the AHP [25] model to cope with the problem, but a detailed discussion of this issue is beyond the scope of this paper.

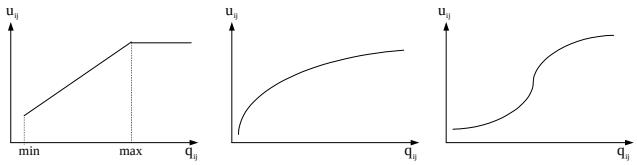

Figure 1. Typical dimension-wise utility functions.

Given the Quality Index, a dimension-wise utility could be defined and hence the application utility. Again, the example task profile is presented in the next subsection to illustrate the possible structure of dimension-wise utility functions and application utility functions.

3.4. Example Dimensional & Application Utilities

Continuing with our example task T_i, assume that T_i is a remote surveillance system where video is much more important to the user than audio. Let's further assume that that SQCIF, gray-scale, low frame rate is fine for video, and there is no need for encryption. Figure 2 depicts an example dimensional utility function for T_i's frame rate. Therefore we could have the following minimum quality specification

$$q_i^{\min} = (1, 1, 2, 1, 1, 1, 2)$$

which corresponds to the following minimum quality

$$(\text{SQCIF}, 1\,\text{bpp}, 2\,\text{fps}, \text{no encryption}, 8\,\text{kHz}, 8\,\text{bps}, 100\,\text{ms}).$$

Since video is more important to the user than audio, an example application utility function for T_i could be:

$$u_i(q_1, \ldots, q_7) = 5\Big(\underbrace{u_{i1}(q_{i1}) + \cdots + u_{i4}(q_{i4})}_{\text{video}}\Big) + 1\Big(\underbrace{u_{i5}(q_{i5}) + \cdots + u_{i7}(q_{i7})}_{\text{audio}}\Big)$$

where video quality is weighted five times more than audio.

3.5. User Interface Consideration

If a user were to choose quality on a scale of 1 to 10 with some pre-determined quality choices preset by the system, the user-interface for specifying utility values can be relatively simple. A more flexible, but also more sophisticated, scheme is to have a set of parameterized utility curves available for each quality dimension, and to have the user pick the curves and instantiate appropriate parameters/coefficients. In our system, the instantiation is carried by letting the user graphically specify *Satisfaction Knee Point parameters*. For the exponential-decay used in the previous example ($u_{i3}(q_{i3}) = 1 - e^{aq_{i3}+b}$), the user could specify the 50% and 95% levels. This is enough to uniquely determine a and b. For example, a user could specify (5fps, 0.50) and (20fps, 0.95), and the corresponding utility curve would then be the one shown in Figure 2, with $a = -0.1535$ and $b = 0.0744$.

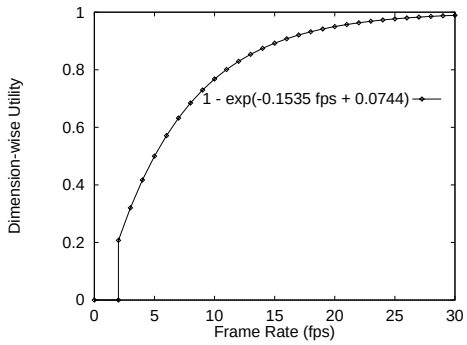

Figure 2. Dimensional utility.

4. Issues on Algorithm Choice & Methodology

4.1. Algorithm Design Issues — Solution Quality vs. Computational Complexity

As is shown in [17] the QoS management optimization problems are NP-hard. As a consequence, there are no optimal solution techniques other than a (possibly complete) enumeration of the solution space. On the other hand, QoS management calls for on-line solutions as the optimization module will ideally be at the heart of an admission control and adaptive QoS management system. Therefore the goal is to strike the right balance between solution quality and computational complexity.

For more than two decades, many researchers from the fields of mathematics, computer science and operations research have been working on the combinatorial optimization and solving NP-hard problems. Three approaches [2] [19] have been well studied and widely used:

Enumerative methods:

that are guaranteed to produce an optimal solution [7][8];

Approximation schemes:
that run in polynomial time [26][9]; and

Heuristic techniques:
(under the general heading of local search) that do not have an *a priori* guarantee in terms of solution quality or running time, but provide a robust approach to obtaining a high-quality solution to problems of a realistic size in reasonable time [2].

An important attribute is the incremental and state-reuse property of a scheme, so as to avoid having to completely redo expensive computations to accommodate the dynamic arrival and departure of tasks. Also, we ensure that all algorithms should be formulated so that the search for an optimal solution can be terminated at any time while still reaching a *feasible*, but sub-optimal and hopefully good, solution. These two properties are essential for an algorithm to be used in an online (or near-online) environment.

We have developed a series of schemes that give an approximation, an approximation with a bound, and an exact solution, each with increased asymptotic computational complexity.

4.2. Resource-Utility Function Composition

Due to the multi-dimensional nature of quality of services, there is often no complete ordering among quality-of-service points, even for individual tasks. So some structural composition is required for those algorithms that call for mapping from resources to utility. Specifically, an R-U (Resource to Utility) function can be constructed for each task through a *task profile*.

Recall that given a resource allocation to a task, one could use the resource to improve different QoS dimensions, which could therefore lead to different utility values. But the most valued QoS point for each resource value can be picked, as intuitively, we certainly want to assign resources to those quality points with the highest utility value.

We therefore define a function $g_i : R \rightarrow \mathbb{R}$, such that

$$g_i(r) = \max\{\, u_i(q) \mid r \models_i q \,\} \qquad (1)$$

and define $h_i : R \rightarrow \mathcal{P}(Q_i)$ to retain the quality points associated with the utility value $g_i(r)$:

$$h_i(r) = \{\, q \in Q_i \mid u_i(q) = g_i(r) \wedge r \models_i q \,\} \qquad (2)$$

Then an R-U graph can be generated for each task, each of which would be a step function (perhaps with multiple level of steps).

5. SRMD Algorithms

5.1. An Approximation Scheme

By constructing the convex hull for each of g_i (see Equation (1)) functions we get piece-wise linear relaxation functions g_i°, $i = 1 \ldots n$. The gradients of g_i° can be used as a heuristic to allocate resources among these tasks[3]. Let

$$C_i = \left\langle \binom{u_{i1}}{r_{i1}}, \ldots, \binom{u_{ik_i}}{r_{ik_i}} \right\rangle$$

be the utility function g_i's discontinuity points in increasing r-order (therefore increasing u-order as well), and we will refer to it as the r-u-pair list. Denote by r^c the current remaining resource capacity after certain resources have been allocated; s_list$[i].t$, s_list$[i].r$, s_list$[i].u$ the task id, the associated r-value and u-value of the corresponding r-u-pair list; $r[i]$ and $u[i]$ are the resource allocated and corresponding utility for T_i respectively.

asrmd1$(n, C_1, \ldots, C_n)$
1. **for** $i = 1$ **to** n **do**
2. $\quad C_i' :=$ **convex_hull_frontier**(C_i)
3. $\quad r[i] := 0$
4. $\quad u[i] := 0$
5. *s_list*= **merge**$(C_1', \ldots, C_n')$
6. $r^c := r^{\max}$
7. $u := 0$
8. **for** $j = 1$ **to** $|$*s_list*$|$ **do**
9. $\quad i :=$ *s_list*$[j].t$
10. $\quad \beta =$ *s_list*$[j].r - r[i]$
11. $\quad$ **if** $(\beta \leq r^c)$ **then**
12. $\quad\quad r^c := r^c - \beta$
13. $\quad\quad r[i] :=$ *s_list*$[j].r$
14. $\quad\quad u[i] :=$ *s_list*$[j].u$ /* *Update alloc of* T_i */
15. $\quad$ **else**
16. $\quad\quad$ **break**
17. **for** $i = 1$ **to** n **do**
18. $\quad q[i] := h_i(r[i])$ /* *See Equation (2).* */
19. $\quad u := u + u[i]$
20. **return** $(q[1], \ldots, q[n], u)$

Note that each $q[i]$ provides a set of quality choices from which T_i (its user, or session manager) could choose to make further QoS tradeoffs.

Notice that in the implementation, we actually replace "break" in line 16 with "continue" (i.e., let the loop continue when condition at step 11 does not hold). This means that after the optimal condition is violated, the residual capacity (r^c) will be greedily filled. The continuation can be thought of as a post-optimization process. The error bound property to be proved below holds for either case.

Let $L = \max_{i=1}^{n} |C_i|$. After the procedure *convex_hull_frontier*[4] (which takes time $O(nL)$) a convex hull frontier with non-increasing slope segments (piece-wise concave) is obtained for each task. The segments are merged at step 5 using a divide-and-conquer approach with

[3]The algorithm is somewhat similar to the one described in [24] but without the restrictions, such as piece-wise concavity and continuity, being assumed.

[4]Overmars & Leeuwen's [21] algorithm, or simply the quickhull [22] or Graham-Scan [6] when C_i are not pre-sorted.

$\log_2 n$ levels, each level has nL comparisons. Merging thus takes time $O(nL \log n)$. Steps 8 through 16 require $O(|s\_list|) = O(nL)$. Steps 17 through 19 take $O(n)$. The total running time of the algorithm is thus $O(nL \log n) + O(nL) = O(nL \log n)$.

Denote by δ_i the maximum utility difference between adjacent discontinuity points of C'_i, i.e., the largest increase in utility for task T_i on the convex hull frontier. Let $\chi = \max_{i=1}^{n} \delta_i$. Denote by U_{opt} the optimal utility and U_{srmd1} the approximation result obtained by algorithm *asrmd1*.

Theorem 1 *U_{srmd1} is within χ of U_{opt}, i.e. $U_{opt} - \chi < U_{srmd1} \leq U_{opt}$.*

Proof Interested readers should see [17].

Although *asrmd1* is a polynomial approximation algorithm with a describable and potentially small error bound from the optimal result, the bound is not controllable. Section 5.3 presents another polynomial approximation scheme with a controllable error bound.

5.2. An Optimal Solution Scheme

Assume that the resources are always allocated *and* used in units of $r^{\max}/P$ for some integer P. If, for example, $P = 100$ this would mean that allocation is in integer percentages. Under this assumption, we can characterize the structure of the optimal solution and recursively define its value as follows:

Denote by $v(i, p)$ the maximum utility achievable when the first i tasks are considered with resource $r^{\max}p/P$ available for allocation, and define

$$v(i,p) = \max_{p' \in \{0,\ldots,p\}} \{g_i(p') + v(i-1, p-p')\} \quad (3)$$

The set of interesting p' values is in fact just all the (starting) discontinuity points of g_i (see Equation (1)). Therefore $v(n, P)$ will be the maximum utility achievable by allocating up to $r^{\max}$ to the n tasks, i.e., the best allocation overall.

Based on Equation 3, the following algorithm *srmd* can be constructed through dynamic programming. Let

$$C_i = \left\langle \begin{pmatrix} u_{i1} \\ r_{i1} \end{pmatrix}, \ldots, \begin{pmatrix} u_{ik_i} \\ r_{ik_i} \end{pmatrix} \right\rangle$$

denote the utility function g_i's discontinuity points in increasing u-order, and *qos*(i, p) the list of QoS allocation choices for T_1 through T_i towards $v(i, p)$.

srmd$(n, P, C_1, \ldots, C_n)$
1. **for** $p = 0$ **to** P **do**
2. *qos*$(0, p) :=$ *nil*, $v(0, p) := 0$
3. **for** $i = 1$ **to** n **do**
4. *qos*$(i, 0) :=$ *nil*, $v(i, 0) := 0$
5. **for** $p = 1$ **to** P **do**
6. $u^* := 0, j^* := 0$
7. **for** $j = 1$ **to** $|C_i|$ **do**
8. **if** $(r_{ij} > p$ **or** $h_i(r_{ij}) < q_i^{\min})$ **break**
9. $u := u_{ij} + v(i-1, p-r_{ij})$
10. **if** $u > u^*$ **then**
11. $u^* := u$
12. $j^* := j$
13. *qos*$(i, p) :=$ *qos*$(i-1, p-r_{ij^*})$ **concat** $[h_i(r_{ij^*})]$
14. $v(i, p) := u^*$
15. **return** $v(n, P)$ and $qos(n, P)$

The result $v(n, P)$, the utility accrued when 100% of the resource is available, is optimal. Let $L = \max_{i=1}^{n} |C_i|$. The time complexity of the algorithm is $O(nLP)$ or $O(nP^2)$, which is pseudo-polynomial.

One of the positive aspects of this scheme (also true for the corresponding MRMD scheme described in [17]) is its incremental and state-reuse property in which when a new task arrives, previous results can be directly reused to avoid the expensive recomputation of the complete new task set. See [17] for detail discussion. The algorithm, with minor change, would be suitable to deal with the *stability* problem when a user prefers (or a policy requires) a relative stable quality. Note that the computation time of srmd increases as the granularity of resource allocation increases. For example, if the required resource allocation is in fractional units, say 0.1%, of the available, then $r^{\max}$ needs to be increased to 1000. Nevertheless, for fixed values of resource granularity and a fixed number of quality levels, the computation time for srmd is polynomial in n, the number of tasks. We will see from our experimental results that srmd is a fairly efficient algorithm for finding the optimal solution for certain resource allocation granularities.

5.3. A Polynomial Scheme with Controllable Bound

The algorithm *asrmd2* to be described will give an approximate quality and resource allocation which is guaranteed to have a maximum relative error, ε, where $0 < \varepsilon < 1$ is a user-specified value. A relative error of ε means that the utility U_{srmd2} found by the algorithm satisfies

$$(1-\varepsilon)U_{\text{opt}} \leq U_{\text{srmd2}} \leq U_{\text{opt}}$$

where U_{opt} is the optimal utility.

Before presenting *asrmd2*, let us define some data structures and operations to be used in the algorithm. All of utility function g_i's discontinuity points are listed in increasing u-order as

$$C_i = \left\langle \begin{pmatrix} u_{i1} \\ r_{i1} \end{pmatrix}, \ldots, \begin{pmatrix} u_{ik_i} \\ r_{ik_i} \end{pmatrix} \right\rangle$$

where $\binom{0}{0}$ is the first element, and referred to as r-u-pair lists. We also define the following operation for r-u-pair lists and r-u-pair elements.

$$\left\langle \begin{pmatrix} u_1 \\ r_1 \end{pmatrix}, \ldots, \begin{pmatrix} u_k \\ r_k \end{pmatrix} \right\rangle + \begin{pmatrix} u \\ r \end{pmatrix} = \left\langle \begin{pmatrix} u_1+u \\ r_1+r \end{pmatrix}, \ldots, \begin{pmatrix} u_k+u \\ r_k+r \end{pmatrix} \right\rangle$$

Note, that this operation produces a new r-u-list that is sorted non-decreasingly in the u-value. From now on such sorting will be assumed.

Let A and B be r-u-pair lists. The procedure *combine_and_merge* will combine A and B into a single r-u-pair list.

combine_and_merge(A, B)
1. **foreach** $b_i \in B$
2. $\quad A_i := A + b_i$ /* A_i *is now increasing in* u*-value.* */
3. $C :=$ **merge**$(A_1, \ldots, A_k)$
4. **return** C.

where $k = |B|$, and A_i, $1 \leq i \leq k$, are intermediate r-u-pair lists.

Steps 1 and 2 take $O(|A|\,|B|)$, step 3 takes $O(|A|\,|B| \log |B|)$ if we do it using divide-and-conquer and merge lists in pairs recursively. So *combine_and_merge* is $O(|A|\,|B| \log |B|)$. The procedure *resource_sieve* trims those r-u-pair elements of list $L = \left\langle \binom{u_{i1}}{r_{i1}}, \ldots, \binom{u_{in}}{r_{in}} \right\rangle$ which do not satisfy $r < r^{\max}$; and those inefficient elements. By inefficient we mean: for each element $\binom{u_i}{r_i}$ and element $\binom{u_{i+1}}{r_{i+1}}$ from L, if $r_{i+1} \leq r_i$ (and $u_i \leq u_{i+1}$ since elements are sorted) then $\binom{u_i}{r_i}$ is inefficient and should be removed from L. Intuitively, we only want to keep those choices that use less resource while achieving the same or higher utility. The procedure takes time $O(|L|)$.

resource_sieve$(L, r^{\max})$
1. $i := 1$
2. **while** $i < |L|$ **do**
3. $\quad$ **if** $r_{i+1} > r^{\max}$ **then**
4. $\quad\quad$ Remove $\binom{u_{i+1}}{r_{i+1}}$ from L
5. $\quad$ **else**
6. $\quad\quad$ **while** $i \geq 1$ **and** $r_{i+1} \leq r_i$ **do**
7. $\quad\quad\quad$ Remove $\binom{u_i}{r_i}$ from L
8. $\quad\quad\quad$ $i := i - 1$
9. $\quad\quad$ $i := i + 1$
10. **if** $r_i > r^{\max}$ **then**
11. $\quad$ Remove $\binom{u_i}{r_i}$ from L
12. **return** L.

The procedure *representative_list* trims the r-u-pair list further in $O(|L|)$ by removing elements that are too close to other element in terms of u-value. That is, for each adjacent $\binom{u_i}{r_i}$ and $\binom{u_{i+1}}{r_{i+1}}$ from L, if $(u_{i+1} - u_i)/u_{i+1} \leq \delta$, then $\binom{u_{i+1}}{r_{i+1}}$ can be presented by $\binom{u_i}{r_i}$ with a discrepancy of at most δ w.r.t. the u-value of $\binom{u_{i+1}}{r_{i+1}}$, and therefore $\binom{u_{i+1}}{r_{i+1}}$ can be removed from L.

representative_list(L, δ)
1. $L' := \left\langle \binom{u_1}{r_1} \right\rangle$
2. $u^* := u_1$
3. **for** i = 2 **to** $|L| - 1$ **do**
4. $\quad$ **if** $(u^* < u_i(1 - \delta))$ **then**
5. $\quad\quad$ append $\binom{u_i}{r_i}$ to L'
6. $\quad\quad$ $u^* := u_i$
7. **return** L'

Given the above procedures, the bounded approximation scheme can be constructed as follows. For the sake of simplicity of the complexity analysis to be followed, we introduce some intermediate lists L_{ia}, L_{ib} and L_i.

asrmd2$(C_1, ..., C_n, \varepsilon)$
1. $L_0 := \left\langle \binom{0}{0} \right\rangle$
2. $\delta := \varepsilon / n$
3. **for** $i = 1$ **to** n **do**
4. $\quad L_{ia} :=$ **combine_and_merge**(L_{i-1}, C_i)
5. $\quad L_{ib} :=$ **resource_sieve**$(L_{ia}, r^{\max})$
6. $\quad L_i :=$ **representative_list**(L_{ib}, δ)
7. let $\binom{u}{r}$ be the element w/ the largest utility value in L_n
8. **return** $\binom{u}{r}$

Without *resource_sieve* and *representative_list* the length of the list obtained at step 4 in *asrmd2* could increase exponentially. We will show that with those steps, the length of of L_i will be bounded by $\left\lfloor \frac{n \ln(u_{\text{up}}/u_{\text{low}})}{\varepsilon} + 2 \right\rfloor$, where u_{up} and u_{low} are easily determined from C_i and f is a suitable constant.

Theorem 2 asrmd2 *is a polynomial approximation for SRMD, and its approximation is within a bound of* ε *w.r.t. the optimal.*

Proof Interested readers could see [17] for detail.

6. Practical Performance Evaluation

In the previous section, we discussed the theoretical behavior of the SRMD algorithms. We will now examine their practical performance. We compare actual computation cost in terms of running time, and solution quality with respect to the optimum.

6.1. Comparative Evaluation of asrmd1 & srmd

Before we review the performance results on the complete set of ranges mentioned above, we present a simple example to illustrate the asrmd1 algorithm. Figure 3 depicts a set of simplified task profiles after the resource-utility structural composition is done (see Equation (1) and (2)). In this case, there are eight tasks, each with twenty different quality levels specified, and a total available resource of 100.

Plotted in Figure 4 are the approximation data points for each task after the **convex_hull_frontier** procedure is called in asrmd1. Table 1 shows the resource allocation results of

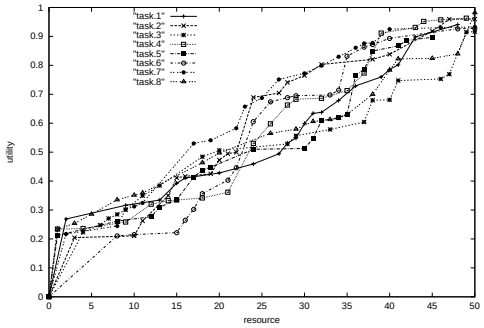

Figure 3. Simplified task profiles.

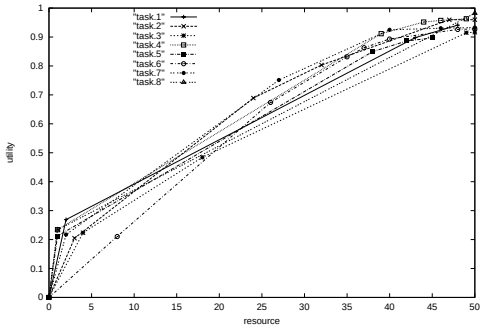

Figure 4. Convex hull frontier approximation.

both asrmd1 and srmd, and they happen to be exactly the same.

We now present a series of experiments conducted to compare the run-time efficiency and solution quality of asrmd1 relative to the optimal srmd algorithm. All experiments were conducted on a 300 MHz Pentium machine running RedHat Linux. The three main variables among the parameters we study are:

- Number of tasks (num_tasks: ranging from 8 to 1024).
- Number of quality levels (ranging from 8 to 128).
- Total available resources ($r^{\max}$: 10^2 to 10^6 units).

Note that the number of quality levels is specified in terms of utility value, which is less than or equal to the number of quality points. The point with the highest utility is taken when the same resource allocation supports multiple quality points.

task	asrmd1		srmd	
	resrc	utility	resrc	utility
1	2	.2689	2	.2689
2	24	.6891	24	.6891
3	18	.4842	18	.4842
4	1	.2342	1	.2342
5	1	.2121	1	.2121
6	26	.6738	26	.6738
7	27	.7513	27	.7513
8	1	.2337	1	.2337
total	100	3.547	100	3.547

Table 1. Example resource allocations.

We can think of $r^{\max}$ in terms of the precision of the resource allocation for the srmd algorithm. When $r^{\max} \geq 1000$, srmd effectively gives fractional resource allocations. For example, $r^{\max} = 10000$ corresponds to a precision of one one-hundredth of resource units. While asrmd1 and asrmd2 handle non-integral resource allocation without any added computational complexity, the computation time of srmd increases as the granularity of resource allocation increases.

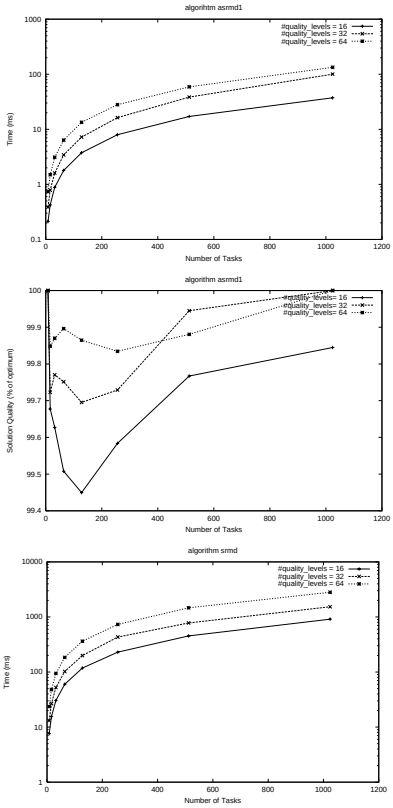

Figure 5. Runtime and quality: asrmd1 / srmd ($r^{\max} = 800$).

Figures 5 and 6 present the run-times of algorithms asrmd1 and srmd. Each figure contains three graphs. The first and third graphs plot the run-times (in ms) for algorithms asrmd1 and srmd respectively as the number of tasks in the system is increased. The second graph plots the solution quality of algorithm asrmd1 relative to the optimal solution obtained by srmd. Figure 5 has $r^{\max} = 800$ while Figure 6 has $r^{\max} = 12800$, each tested with various quality levels — 16, 32, 64, or 128. Each workload in these experiments was generated 100 times, each with different task profiles so that we could examine the solution quality of approximation algorithms in a broad range of scenarios.

Notice how algorithm asrmd1 consistently runs about an order of magnitude faster than the exact algorithm srmd in Figures 5 and 6. The difference approaches two orders of magnitude when the granularity of resource allocation is finer in Figures 5 and 6. Notice further that the average solution quality for algorithm asrmd1 in the second graph of each figure stays above 99% for most cases. Since the plotted values are the average over 100 runs, the worst case obviously is lower. However, in general, it is easy to con-

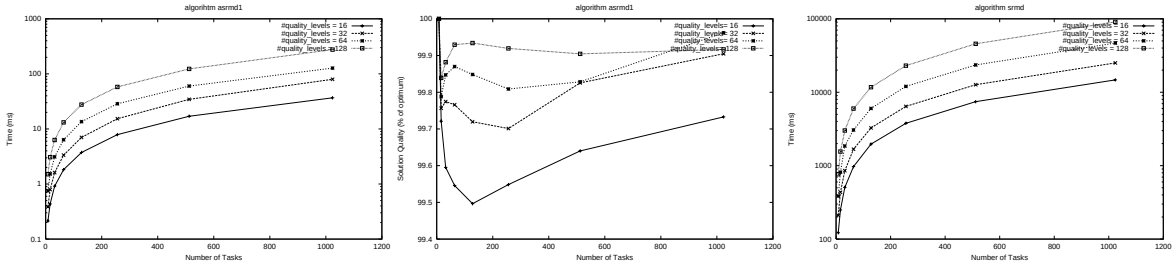

Figure 6. Runtime and quality: asrmd1 / srmd ($r^{\max} = 12800$).

clude that the approximation algorithm asrmd1 exhibits excellent behavior in achieving near-optimal results within a small fraction of time needed to find the optimal solution.

6.2. Comparative Evaluation of asrmd1 & asrmd2

We conducted a second series of experiments to compare the performance of algorithms asrmd1 and asrmd2. To save time and space, the number of QoS options per task was held constant at 16 in each experiment, and ε was chosen to be a constant 0.01 (i.e. the desired quality obtained by asrmd2 must be within 1% of the optimal solution).

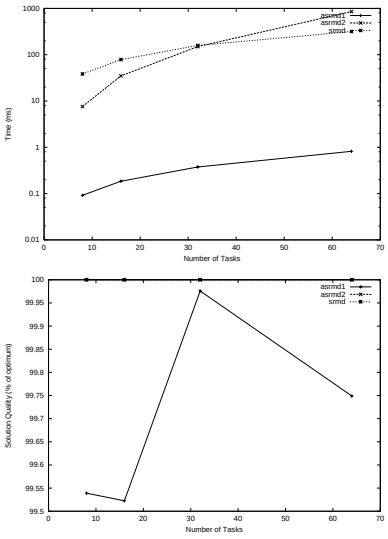

Figure 7. asrmd1 / asrmd2 / srmd ($r^{\max} = 10^4$).

The run-times and solution qualities of the two approximation algorithms along with the optimal srmd algorithm were measured with $r^{\max} = 10^4$, 10^5 and 10^6. The resulting graphs are plotted in Figure 7 and 8 (due to the limited space, only the 10^4 and 10^6 cases are shown). The first of two graphs in each figure plots the run-times of the three algorithms as the number of tasks is increased (to keep run-times feasible, the maximum number of tasks tested had to be significantly dropped). The second graph plots the solution qualities.

As discussed earlier, algorithm asrmd2 is very promising from a theoretical point of view: it always delivers a guaranteed solution quality in polynomial time. Unfortunately its actual running time, as shown in Figure 7 and 8, is up to two orders of magnitude of asrmd1. The solution quality graphs plot the solution quality of algorithms asrmd1 and asrmd2. They show that asrmd1 is mostly within 1% of the optimal solution while asrmd2, which must always be within 1%, on the average yields a solution very close to the optimal solution. However, the difference in run-times is much too high for asrmd2, particularly when the solution quality obtained by asrmd1 is extremely good.

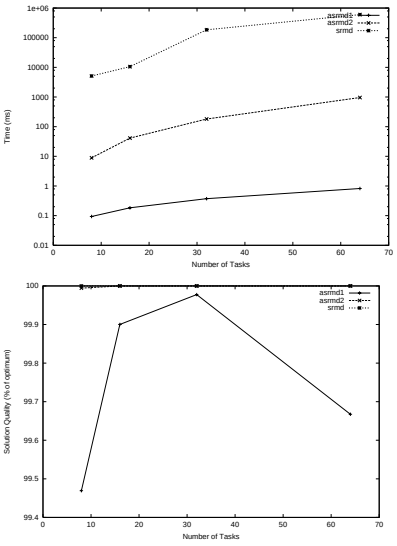

Figure 8. asrmd1 / asrmd2 / srmd ($r^{\max} = 10^6$)

Based on the above two sets of experiments, we conclude that the asrmd1 algorithm using the convex hull frontier approach yields the biggest benefit for the limited computational time that it consumes. It is practically useful to note that in absolute terms, even with 128 tasks and 128 quality levels per task, asrmd1 yields a near-optimal result in about 20ms. The result is also within 0.5% of the optimal solution on the average. Such runtime can be used in practice in real-time systems to make near-optimal online QoS-based allocations.

7. Conclusion

We have proposed a translucent[5] QoS management framework [15][23] for QoS optimization in systems that must satisfy application needs along multiple dimensions The architecture consists of a semantically rich (in terms of

[5]The framework is translucent in the sense that some aspects are made visible to the end-users so that they can control the delivered QoS parameters, while at the same time hiding how the requested delivery is accomplished.

customizability and expressiveness) QoS specification interface for multi-dimensional QoS provisioning, a quality-of-service index model to help the user make quality trade-off decision, and a unified QoS-based admission control and resource planning system. Our QoS specification allows applications and users to put values on the different levels of service that the system can provide. When "value" is taken literally, this means that our model is able to facilitate market-efficient resource distribution.

We presented and compared one exact and two near-optimal algorithms with one yielding a solution within a bounded distance from the optimal solution and another yielding a solution within a user-specified distance from the optimal solution. Detailed evaluations of the run-times of the three algorithms and their solution qualities shows that the first near-optimal algorithm performs very close to the optimal solution. It also has very practical run-times that it can even be used on-line.

8. Acknowledgement

The authors would like to thank Professor John Hooker at the Graduate School of Industrial Administration at Carnegie Mellon for many insightful discussions. Our thanks are also due to Julia Deems, Christos Faloutsos, Nelu Mihai, John Wilkes and members of the Amaranth project.

References

[1] Amaranth Project. Amaranth White Paper, Dec. 1996.

[2] E. Aarts and J. Lenstra, editors. *Local Search in Combinatorial Opitmization*. John Wiley & Sons, 1997.

[3] E. Atkins, T. Abdelzaher, and K. Shin. QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. In *Proceedings of the IEEE Real-time Technology and Applications Symposium*, June 1997.

[4] J. Bolliger and T. Gross. A Framework-Based Approach to the Development of Network-Aware Applications. In *IEEE Trans. Software Engineering*, volume 24, May 1998.

[5] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proceedings of the SIGCOMM '92*, pages 14–26, Oct. 1992.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press / McGraw-Hill, 1990.

[7] T. Ibaraki. Enumerative Approaches to Combinatorial Optimization-I. *Annals of Operations Research*, 10, 1987.

[8] T. Ibaraki. Enumerative Approaches to Combinatorial Optimization-II. *Annals of Operations Research*, 11, 1987.

[9] O. Ibarra and C. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of ACM*, 22:463–468, 1975.

[10] ITU. ITU-T Recommendation H.263 - Video Coding for Low Bit Rate Communication, July 1995.

[11] K. Jeffay, D. Stone, and F. Smith. Kernel support for live digital audio and video. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 10–21, Nov. 1991.

[12] E. D. Jensen, C. D. Locke, and H. Tokuda. A Time-Driven Scheduling Model for Real-Time Operating Systems. In *Proceedings of the 6th IEEE Real-Time Systems Symposium*, Dec. 1985.

[13] M. Jones, P. Leach, R. Draves, and J. Barrera. Modular Real-Time Resource Management in the Rialto Operating System. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, May 1995.

[14] K. Kawachiya and H. Tokuda. A Negotiation-Based Resource Management Framework for Dynamic QoS Control. In *Proceedings Real-Time Mach Workshop '97*, Aug. 1997.

[15] C. Lee. A Translucent QoS Architecture. In *Proceedings of the RT-Mach Workshop'97*, Aug. 1997.

[16] C. Lee, Y. Katsuhiko, R. Rajkumar, and C. Mercer. Predictable Communication Protocol Processing in Real-Time Mach. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 1996.

[17] C. Lee and D. Siewiorek. An Approach for Quality of Service Management. Technical Report CMU-CS-98-165, Computer Science Department, CMU, Oct. 1998.

[18] C. D. Locke. *Best-effort Decision Making for Real-Time Scheduling*. PhD thesis, CMU, 1986.

[19] S. Martello and P. Toth. *Knapsack Problems – Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[20] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. In *Proc. ACM Multimedia'95*, Nov. 1995.

[21] M. Overmars and J. Leeuwen. Maintenance of Configurations in the Plan. In *Journal of computer and System Sciences*, volume 23, pages 166–204, 1981.

[22] F. Preparata and M. Shamos. Computational Geometry : An Introduction. In *Texts and Monographs in Computer Science*. Springer-Verlag, 1985.

[23] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A QoS-based Resource Allocation Model. In *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1997.

[24] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical Solutions for QoS-based Resource Allocation Problems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1998.

[25] T. Saaty. Multicriteria Decision Making - The Analytic Hierarchy Process. Technical report, University of Pittsburgh, RWS Publications, 1992.

[26] S. Sahni. Approximation algorithms for the 0-1 knapsack problem. In *Journal of ACM*, volume 23, 1975.

[27] K. Sayood. *Introduction to Data Compression.* Morgan Kaufmann Publishers, Inc., 1996.

[28] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, pages 8–18, Sept. 1993.