Practical Solutions for QoS-based Resource Allocation Problems*

Ragunathan (Raj) Rajkumar, Chen Lee, John P. Lehoczky†, Daniel P. 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: The QoS-based Resource Allocation Model (Q-RAM) proposed in [20] presented an analytical approach for satisfying multiple quality-of-service dimensions in a resource-constrained environment. Using this model, available system resources can be apportioned across multiple applications such that the net utility that accrues to the end-users of those applications is maximized. In this paper, we present several practical solutions to allocation problems that were beyond the limited scope of [20]. First, we show that the Q-RAM problem of finding the optimal resource allocation to satisfy multiple QoS dimensions (at least one of which is dependent on another) is NP-hard. We then present a polynomial solution for this resource allocation problem which yields a solution within a provably fixed and short distance from the optimal allocation. Secondly, [20] dealt mainly with the problem of apportioning a single resource to satisfy multiple QoS dimensions. In this paper, we study the converse problem of apportioning multiple resources to satisfy a single QoS dimension. In practice, this problem becomes complicated, since a single QoS dimension perceived by the user can be satisfied using different combinations of available resources. We show that this problem can be formulated as a mixed integer programming problem that can be solved efficiently to yield an optimal resource allocation. Finally, we also present the run-times of these optimizations to illustrate how these solutions can be applied in practice. We expect that a good understanding of these solutions will yield insights into the general problem of apportioning multiple resources to satisfy simultaneously multiple QoS dimensions of multiple concurrent applications.

1. Introduction

Several applications have the ability to provide better performance and quality of service if a larger share of system resources is made available to them. Such examples abound in many domains. Feedback control systems can provide better control at higher rates of sampling and control actuation. Multimedia systems using audio and video streams can provide better audio/video quality at higher resolution and/or very low end-to-end delays. Tracking applications can track objects at higher precision and accuracy if radar tracks are generated and processed at higher frequencies. In many cases, computationally intensive algorithms can provide better results than their less-demanding counterparts. Even interactive systems can provide excellent response times to users if more processing and I/O resources are made available. Conversely, many applications can still prove to be useful and acceptable in practive even though the resources needed for their maximal performance are *not* available. 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.

The QoS-based Resource Allocation Model (Q-RAM) proposed in [20] addressed the following question: "How does one allocate available resources to multiple concurrent applications?". This question was posed in the context where applications can operate at high levels of quality or acceptably lower levels of quality based on the resources allocated to them. The novelty of Q-RAM is that it allows multiple Quality of Service requirements such as timeliness, cryptography and reliable data delivery to be addressed and traded off against each other.

In this paper, we address some significant and open problems posed by Q-RAM. For example, much of the QoS work focuses on allocating a single time-shared resource such as network bandwidth. In real-time and multimedia 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. The solutions that we provide turn out to be very efficent to be used in practice.

1.1. Q-RAM: The QoS-based Resource Allocation Model

We now provide a brief overview of Q-RAM referring the reader to [20] for more details. The goal of Q-RAM is to ad-

*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.

dress two problems:

- Satisfy the simultaneous requirements of multiple applications along multiple QoS dimensions such as timeliness, cryptography, data quality and reliable packet delivery, and
- Allow applications access to multiple resources such as CPU, disk bandwidth, network bandwidth, memory, etc. simultaneously.

Q-RAM uses a dynamic and adaptive application framework where each application requires a certain minimum resource allocation to perform acceptably. An application may also improve its performance with larger resource allocations. This improvement in performance is measured by a *utility function*.

Q-RAM considers 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 QoS dimension such as timeliness, security, data quality and dependability.
- An application may require access to multiple resource types such as CPU, disk bandwidth, network bandwidth and memory.
- 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.

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 $\sum_{i=1}^{n} R_{i,j} \leq \mathbf{R}_j$. The following definitions are used:

- 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 utilities, 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 for 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 $R_{i,j}^{min} = \sum_{k=1}^{d} R_{i,j}^{\min_k}, 1 \leq j \leq m$.

The assumptions of Q-RAM that are used in this paper are:

- The applications are independent of one another.
- The available system resources are sufficient to meet the minimal resource requirements of each application on *all* QoS dimensions.
- 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.

Resource allocation schemes presented in [20] assumed a single resource and thereby solved only the simpler problem dealing with multiple QoS dimensions. Resource allocation schemes in the presence of multiple resources were beyond its scope but are the subject of Section 3 in this paper.

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

Each application has a relative weight w_i, but this can be ignored in the optimization step without loss of generality (by scaling an application's utility values by the same weight).

[1]This aspect is stated as a simplification; by choosing different implementation schemes, the minimum requirement on a resource may change. Please see Section 3.

1.3. Related Work

Significant research has been carried out for making resource allocations to satisfy specific application-level requirements. The domain of operations research in particular has spawned over several decades multiple resource allocation problems. For example, [8] characterizes several problems and formally summarizes known solutions to these problems. Other 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, [6] studies the problem of how to allocate network packet processing capacity assuming bursty traffic and finite buffers. In [10], the problem of the establishment of real-time communication channels is studied as an admission control problem. The Spring Kernel [24] uses on-line admission control to guarantee essential tasks upon arrival.

With the advent of asynchronous transfer mode networks and their deployment by telephone companies, Quality of Service in terms of bandwidth and timeliness guarantees has been studied in depth (e.g. see [2, 3]). Like most research on real-time scheduling theory [16, 11], QoS research in networks focus on a single QoS dimension (like timeliness or bandwidth guarantees). In addition, network and real-time scheduling research have typically focused on a *single* resource type like network bandwidth and CPU cycles respectively. In contrast, we focus on multiple QoS dimensions spanning timeliness, cryptography, and application quality requirements. We also focus on making resource allocation decisions across multiple resource *types* including processor compute cycles, network bandwidth, and disk bandwidth.

Various system-wide schemes have been studied to arbitrate resource allocation among contending applications. In [1], 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 [9] 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 [15]. Applications are ranked according to their semantic importance, and different adjustment policies are used to obtain or negotiate a particular resource allocation. Q-RAM can also be considered to be a broad generalization of [17] and [23]. A multi-dimensional QoS problem from a tracking perspective is summarized in [12].

1.4. Organization of the Paper

The rest of this paper is organized as follows. In Section 2, we show that the Q-RAM problem of finding the optimal resource allocation to satisfy multiple QoS dimensions (at least one of which is dependent on another) is NP-hard. We then present a polynomial solution for this resource allocation problem which yields a solution within a provably fixed and short distance from the optimal allocation. In Section 3, we study the problem of apportioning multiple resources to satisfy a single QoS dimension. This problem becomes complicated, since resources can be traded off against each other and still yield the same utility. We formulate this problem as a mixed integer programming problem that can be solved efficiently to yield an optimal resource allocation. In Section 4, we present our concluding remarks and discuss problems that remain unsolved.

2. Multiple QoS Dimensions and A Single Resource

The work reported in [20] provided optimal and near-optimal resource allocation schemes for applications which need a single resource, but need to satisfy one or more QoS dimensions. Specifically, optimal resource allocation schemes were provided for the two cases of

1. a single resource and a single QoS dimension, and
2. a single resource and multiple independent QoS dimensions.

Finally, a greedy and sub-optimal algorithm was proposed for the case of a single resource and two or more QoS dimensions, where one QoS dimension is discrete and dependent on another.[2]

In this section, we further study this latter problem of a single resource and a dependent and discrete QoS dimension. We refer to this problem as DDQSRP (Dependent Discrete Dimension and Single Resource Problem). For the sake of simplicity and practicality, we assume that the utility curve is linear from R_i^{min} to a maximum resource requirement R_i^{max} beyond which it becomes flat. An example 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 1. The first or lowermost curve represents the utility curve for the application when a base scheme (such as no encryption) is used. The second curve represents the utility curve when (say) encryption using 40 bits is used. Additional utility is accrued by the use of encryption, which results in a positive vertical offset at the beginning of the curve. At the same time, additional compute cycles are expended to perform encryption resulting in the lateral shift of the curve to the right. In addition, as data volume increases with improved QoS on the base dimension, the encryption cost increases correspondingly. This results in a "flattening" with respect to the curve of the base scheme. This flattening effect is even more pronounced for the third curve, corresponding to (say) encryption using 128 bits.

[2] 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.

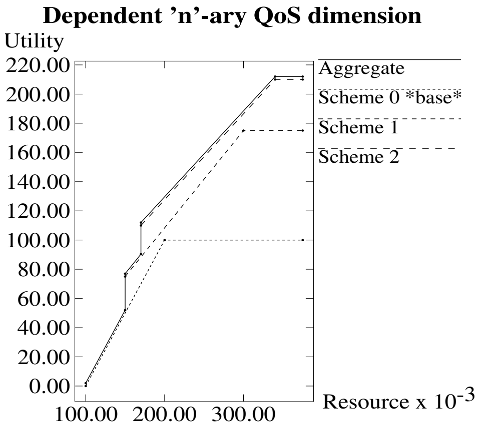

Figure 1. One 3-ary QoS dimension w/ min-linear-max.

We now show that the general problem of finding an optimal resource allocation for applications with these non-concave piecewise linear utility functions is NP-hard.

Theorem 1 *Finding the optimal resource allocation with dependent and discrete QoS dimensions is NP-Hard.*

Proof: We prove this by showing that the inexact 0-1 knapsack problem which is known to be NP-hard (see for example [18]) maps directly to a special case of our QoS problem. The inexact 0-1 knapsack problem is as follows:

Maximize $\sum_{i=1}^{n} v_i$

Subject to:

$$\sum_{i=1}^{n} R_i \leq \mathbf{R}$$

where there are n objects each with size R_i and value v_i, and $\mathbf{R}$ is the size of the knapsack.

The above problem is identical to the following special case of our problem. Suppose each of the users in our QoS allocation has a utility function which is a step function with a single discontinuity, that is

$$U_i(r) = \begin{cases} 0 & \text{if } r < R_i \\ v_i & \text{if } r \geq R_i. \end{cases}$$

Thus user i receives a utility value of v_i if and only if R_i units of additional resource are allocated. A total of $\mathbf{R}$ additional units of resource are available for allocation.

An identity transformation makes the inexact 0-1 knapsack problem a special case of our QoS problem. □

We now provide a polynomial algorithm which yields a solution for this resource allocation problem within a provably fixed and short distance from the optimal allocation. This algorithm uses the "natural" boundary of the set of discontinuities in these utility functions. This boundary is the smallest concave function lying on or above the aggregate utility function. We shall refer to this boundary as the "concave majorant".

2.1. Convex Hulls, Majorants and the Q-RAM Utility Function

The *convex hull* of a set of points in the plane is the smallest convex polygon containing all the points (for example, see [22]). Several algorithms, such as the Graham's Scan [22], Jarvis' March [7], etc., determine the convex hull of a set of points. In our case, the concave majorant is a subset of the path on the convex hull of the points on the aggregate utility curve starting from the minimum resource allocation point to a point with the highest achieved value of utility. Some examples of these concave majorants are presented in Figure 2. Minor variations of the convex hull algorithms can be used to determine the needed concave majorants as follows:

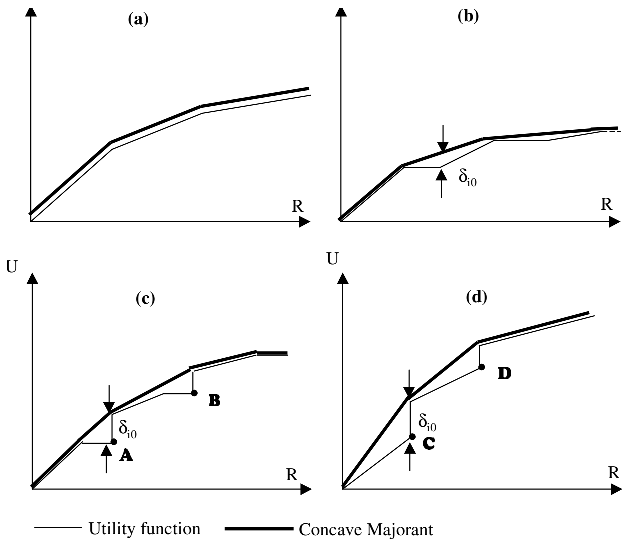

Figure 2. Examples of Concave Majorants for Q-RAM Utility Curves. δ_{i0} represents the maximum vertical separation of the convex majorant from application τ_i's aggregate utility curve.

Consider the discontinuity points *A, B, C, D, E, F, G, H and I* corresponding to a utility function similar to the one in Figure 2-(c) plotted separately in Figure 3-(a) and 3-(b).

Jarvis' March. Figure 3-(c) illustrates how the variant of the Jarvis' March works. Start from the minimum resource allocation point (A) which is known to be in our path. From there, travel to the point with the weakest right turn (highest slope) drawing line segment 1 shown in 3-(c). Repeat until we run out of points, drawing line segments

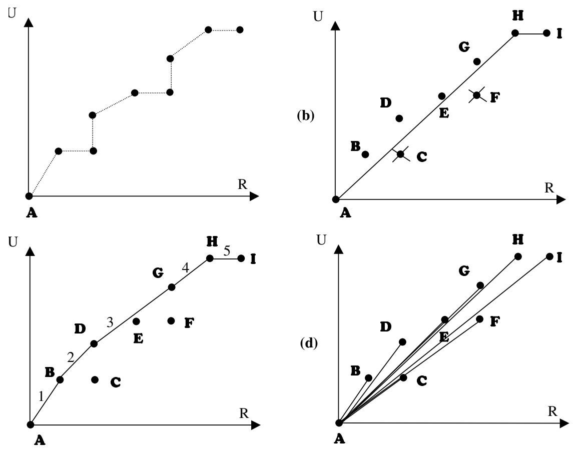

Figure 3. Using Jarvis' March (c) and Graham's Scan (d) for Determining Concave Majorants.

2, 3 4 and 5. This algorithm is conceptually simple but is of complexity $O(n^2)$.

Graham's Scan. Figure 3-(d) illustrates how Graham's Scan works. Again, start from the minimum resource allocation point A which is known to be in our path. Draw lines from this point to all other points and sort them according to their slopes. Travel to B, the point along the highest slope. From there, travel to D, the point with the next highest slope. If ABD is not concave, backtrack to the previous point where concavity is still true and draw a line from there to D; else, continue. Travel to the next unvisited point with the highest slope. Backtracking when necessary, repeat until either no points remain or a maximal point is reached. This algorithm is of complexity $O(n \text{ log}n)$ in general and $O(n)$ with pre-sorted points.

Both Jarvis' March and the Graham's Scan algorithms described above can be optimized further under Q-RAM as shown in Figure 3-(b). Draw a line from the starting point (A) to one of the maximal points (H is preferable but I is acceptable as well). Any points that lie below this line can be ignored by both of the above schemes.

2.2. Polynomial Concave Majorant Optimization Algorithm for DDQSRP

1. For each application, determine the convex hull of the points on its utility function.

2. Pick the subset of the path on the convex hull starting from the minimum resource allocation point to the maximal resource allocation point. That is, obtain the concave majorant of the aggregate utility function.

3. Run the algorithm as described in [20] using the Kuhn-Tucker conditions (see [19], chapter 5): Allocate the available resource to each application in decreasing order of the slope of the concave majorant until the resource runs out or all applications obtain their maximum useful resource allocation. This yields an optimal resource allocation based on the concave majorants.

The determination of the concave majorant has a computational complexity of $O(n)$ (or $O(n \text{ log}n)$ if the discontinuity points are not pre-sorted). The resource allocation algorithm has a computational complexity of $O(k \text{ log}k)$ where $k = nm$ where n is the number of applications and m is the number of discontinuities in each concave majorant.

We now prove that the above algorithm yields a resource allocation that is within a short distance of the optimal resource allocation.

Remark: The Q-RAM utility model assumes that the utility gains due to the introduction of an n-ary and dependent QoS dimension are also themselves non-decreasing and concave.

Notation: Let the maximum vertical separation of the convex majorant from application τ_i's aggregate utility curve be δ_i.

Notation: Let $\delta_0 = max_{1 \leq i \leq n} \delta_i$.

Theorem 2 *The maximum deviation from the optimal resource allocation for DDQSRP using the polynomial concave majorant algorithm is* δ_0.

Proof:

The proof is based on the properties of the concave majorant. First, the vertex points on a concave majorant are points from the original point set. Therefore, in our case, each of the discontinuity points on the concave majorant also corresponds to a discrete point on the aggregate utility function.

Second, by definition, the concave majorant is piece-wise concave. The Kuhn-Tucker conditions are used to allocate the available resource first to the application with the maximum slope at its allocation point. When this is carried out on piece-wise concave functions, all but *at most one* of the applications will be at one of the discontinuity points [20].

Finally, the application of the Kuhn-Tucker conditions leads to an optimal resource allocation if the concave majorants are treated as the actual utility curves. Let the total utility obtained on the concave majorants be $U_{majorant}$. Let the actual total utility obtained by the applications be U_{actual}. Let the total utility obtained by an optimal allocation on the actual utility curves be $U_{optimal}$. Since at most one resource allocation point may lie above an actual utility curve, and the concave majorants lie at or above the actual utility curves, we have,

$$U_{majorant} \geq U_{actual}$$

Since the concave majorants lie at or above the actual utility curves, and $U_{majorant}$ is optimal for the concave majorants, we must have

$$U_{majorant} \geq U_{optimal}$$

We therefore have,

$$U_{majorant} \geq U_{optimal} \geq U_{actual}$$

The largest difference between the utility obtained on the concave majorants and the actual utility occurs when U_{actual} is separated from $U_{optimal}$ by the maximal vertical distance between the concave majorant and an actual utility curve. The maximal separation between an aggregate utility function and its concave majorant cannot be larger than δ_0. The theorem follows. $\square$.

It is possible that δ_0 can be large in some systems, since the inclusion of a scheme (such as cryptography to encrypt sensitive data) may yield significant utility for an application. In such cases, it may seem acceptable to treat the inclusion of the scheme as the minimum requirement for the application. Correspondingly, δ_0 can be reduced. Even if this were not possible, it is encouraging to note that a simple polynomial algorithm yields a resource allocation that is close to the optimal resource allocation, which would take an exponential amount of time to find.

A brute-force exponential algorithm to find the exact optimal resource allocation for DDQSRP is outlined in [21].

3. Single QoS Dimension and Multiple Resources

In this section, we will study the problem of apportioning shares of multiple resources to multiple concurrent applications each of which has to satisfy only a single QoS dimension. We will first motivate the problem and define the solution space available to a resource allocation scheme which is meant to solve this problem.

3.1. The Multiple Resource Problem

Consider an audio-conferencing application where an audio stream is being transmitted from a source to a destination. The stream must be processed at the source processor, transmitted across the network (across one or more hops) and must then be processed again by the destination for playback on a speaker. For the audio stream to be processed on time, all elements on its path would require adequate resources. The sampling rate of the audio stream is a QoS dimension of interest in this context. If sufficient resources are available on the end-processors and the intermediate network elements, a higher audio sampling rate is desirable. However, beyond a CD-Quality sampling rate, any increased utility to the users is very marginal. In other words, this QoS dimension fits the requirements of Q-RAM.

Consider multiple such streams in the system (such as an IP telephony service provider). The goal of Q-RAM is to allocate available resources to each of these streams such that the overall accrued utility is maximized. This problem may seem straightforward and a direct converse of the multiple QoS dimension/single resource problem solved earlier in Section 2. However, one key aspect of this problem tends to be complicated and is discussed next.

3.2. Resource Trade-Offs

Tradeoffs across resources are possible. If processing resources are (relatively) scarce, and network bandwidth is plentiful, the source node may transmit the data in raw form consuming less processor cycles and more network bandwidth. Conversely, if processing resources are relatively plentiful and network bandwidth meager, available processor cycles can be used to compress the data and thereby consume less of the network bandwidth. Different compression schemes also make different tradeoffs with more compression time spent leading to higher compression ratios.

Consider the application utility function defined in Figure 4-(a). As the sampling rate increases, utility improves but beyond a point, it saturates and flattens. For the sake of illustration, consider only two resources (CPU cycles and network bandwidth) and two schemes (with and without compression). In addition, assume that the compression is lossless such that there is no perceived QoS difference to the user whether compression is used or not. The resource consumption functions for this data stream on the CPU and network bandwidth resources are plotted in Figure 4-(b). If no compression is used, demand on the CPU is less and demand on the network bandwidth is higher. The converse is true if compression is used. In either case, the demands on the two resources increase as the sampling rate is increased.

We assume in the rest of this section that the resource consumption functions are linear. As a result, the resource utility functions which combine the application utility function and the resource consumption functions are as plotted in Figure 4-(c). It is informative to note that points marked (1) on the resource utility functions are chosen in tandem; that is, if point (1) is chosen on the CPU utility function, point (1) *must* be chosen on the network bandwidth utility function. The same holds true for points (2), (3) and (4) respectively (and all other pairs of points inbetween).

In general, a total of M schemes is assumed to be available, with $M = 2$ in the example of Figure 4.

3.3. Notation

We shall use the following notation in this section.

Notation: $\mathbf{R}_j$ is the amount available on resource j.

Notation: a_{ik}^j represents the minimum resource demand for application τ_i on resource j when scheme k is used to satisfy the application requirements.

Notation: b_{ik}^j represents the additional resource demand (beyond the minimum requirement of a_{ik}^j) that application τ_i can fruitfully use on resource j when scheme k is in use. In other words, if application τ_i were allocated more than $a_{ik}^j + b_{ik}^j$ amount of resource j, no additional increase in utility will accrue.

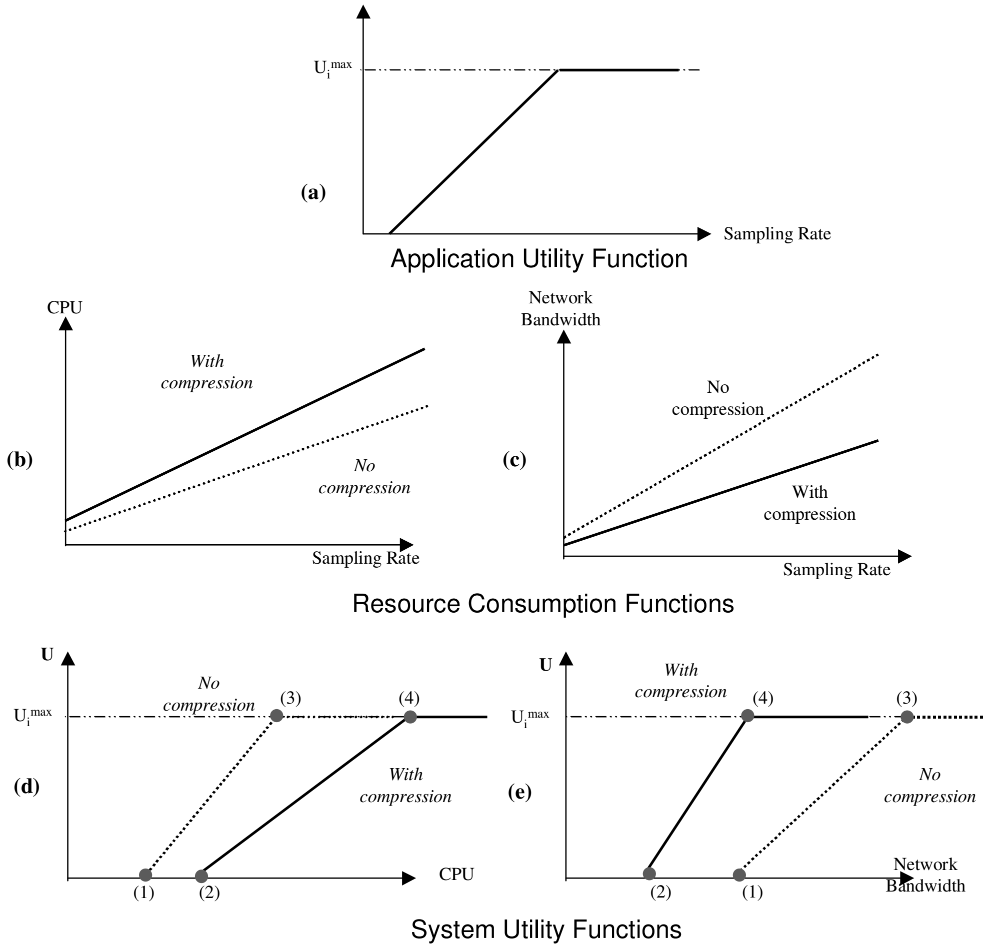

Figure 4. Resource Tradeoffs and Its Impact on Utility Functions.

Notation: u_{ik} is the maximum utility accrued by application τ_i when scheme k is used.

3.4. The Case of $M = 4$

Let (X_i, Y_i) represent $M = 4$ possible schemes for achieving a given quality of service. For example, consider the following. Data is transmitted as is (no compression) and 3 possible compression schemes using different amounts of resources and thereby generating different amounts of data.

Each application τ_i, $1 \leq i \leq n$, has an associated maximal value u_{ik}, $1 \leq k \leq 4$, when scheme k is chosen. Let F_i be the fraction of τ_i's useful range of resource allocation (above its minimal allocation), with $0 \leq F_i \leq 1$.[3]

[3] That is, if τ_i is allocated a total r_i^j of resource $\mathbf{R}_j$ under scheme k, for any k, $F_i = (r_i^j - a_{ik}^j) \div b_{ik}^j$. Q-RAM assumes that $r_i^j \geq a_{ik}^j$ for some value of k.

Objective: maximize

$$\sum_{i=1}^{n}(u_{i1}F_iX_iY_i \ + \ u_{i2}F_i(1-X_i)Y_i$$
$$+u_{i3}F_iX_i(1-Y_i) + u_{i4}F_i(1-X_i)(1-Y_i))$$

subject to:

$$0 \leq F_i \leq 1 \text{ and resource constraints.}$$

Let $1 \leq j \leq m$ index resources, and $X_i, Y_i = 0$ or 1. Since $\mathbf{R}_j$ is the amount available on resource j, the resource constraints can be stated as follows.

Resource Constraints:

$\forall j, 1 \leq j \leq m,$

$$\sum_{i=1}^{n}(a_{i1}^{j} + b_{i1}^{j}F_i)X_iY_i + \sum_{i=1}^{n}(a_{i2}^{j} + b_{i2}^{j}F_i)(1 - X_i)Y_i + \sum_{i=1}^{n}(a_{i3}^{j} + b_{i3}^{j}F_i)X_i(1 - Y_i) + \sum_{i=1}^{n}(a_{i4}^{j} + b_{i4}^{j}F_i)(1 - X_i)(1 - Y_i) \;\leq\; \mathbf{R}_j$$

Expanding, the j^{th} resource constraint becomes

$$\sum_{i=1}^{n} a_{i4}^{j} \;+\; \sum_{i=1}^{n} b_{i4}^{j}F_i + \sum_{i=1}^{n}(a_{i1}^{j} - a_{i2}^{j} - a_{i3}^{j} + a_{i4}^{j})X_iY_i + \sum_{i=1}^{n}(b_{i1}^{j} - b_{i2}^{j} - b_{i3}^{j} + b_{i4}^{j})X_iY_iF_i + \sum_{i=1}^{n}(a_{i3}^{j} - a_{i4}^{j})X_i \;+\; \sum_{i=1}^{n}(a_{i2}^{j} - a_{i4}^{j})Y_i + \sum_{i=1}^{n}(b_{i3}^{j} - b_{i4}^{j})X_iF_i \;+\; \sum_{i=1}^{n}(b_{i2}^{j} - b_{i4}^{j})Y_iF_i \;\leq\; \mathbf{R}_j$$

The objective function and these resource constraints are non-linear in $\{F_i\}$, $\{X_i\}$ and $\{Y_i\}$ containing terms of the form X_iF_i, Y_iF_i, $X_iY_iF_i$ and X_iY_i. These constraints can be linearized by introducing substitutions and adding extra (linear) constraints to the problem[4].

We use the following substitutions:

$$X_iY_i = Z_i$$

$$X_iF_i = f_i$$

$$Y_iF_i = g_i$$

$$Z_iF_i = h_i$$

Note that $X_iY_iF_i = Z_iF_i = h_i$.

The objective function now becomes

$$\sum_{i=1}^{n}(u_{i4}F_i + (u_{i1} - u_{i2})Z_i \;+\; (u_{i2} - u_{i4})f_i + (u_{i3} - u_{i4})g_i \;+\; (u_{i4} - u_{i3})h_i)$$

which is linear in the new variables, and we insert the following additional constraints:

$$\begin{aligned} &f_i \geq 0\\ &g_i \geq 0\\ &h_i \geq 0\\ &f_i - F_i \leq 0\\ &g_i - F_i \leq 0\\ &h_i - Z_i \leq 0\\ &-f_i + F_i + X_i \leq 1\\ &-g_i + F_i + Y_i \leq 1\\ &-h_i + F_i + Z_i \leq 1.\end{aligned}$$

We also need to enforce

$$X_iY_i = Z_i$$

where X_i, Y_i, Z_i are *binary* 0-1 variables. This can be done by adding the constraints

$$\begin{aligned} &Z_i - X_i \leq 0\\ &Z_i - Y_i \leq 0\\ &X_i + Y_i - Z_i \leq 1\end{aligned}$$

The optimal resource allocation problem we call the Multi-Resource Single QoS dimension Problem (MRSQP) thus reduces to the following.

Multi-Resource Single QoS Dimension Problem (MRSQP):

Objective: maximize

$$\sum_{i=1}^{n}(u_{i4}F_i + (u_{i1} - u_{i2})Z_i \;+\; (u_{i2} - u_{i4})f_i + (u_{i3} - u_{i4})g_i \;+\; (u_{i4} - u_{i3})h_i)$$

subject to:

$\forall j, 1 \leq j \leq m,$

$$\sum_{i=1}^{n} a_{i4}^{j} \;+\; \sum_{i=1}^{n} b_{i4}^{j}F_i \;+\; \sum_{i=1}^{n}(a_{i1}^{j} - a_{i2}^{j} - a_{i3}^{j} + a_{i4}^{j})Z_i + \sum_{i=1}^{n}(b_{i1}^{j} - b_{i2}^{j} - b_{i3}^{j} + b_{i4}^{j})h_i + \sum_{i=1}^{n}(a_{i3}^{j} - a_{i4}^{j})X_i \;+\; \sum_{i=1}^{n}(a_{i2}^{j} - a_{i4}^{j})Y_i + \sum_{i=1}^{n}(b_{i3}^{j} - b_{i4}^{j})f_i \;+\; \sum_{i=1}^{n}(b_{i2}^{j} - b_{i4}^{j})g_i \;\leq\; \mathbf{R}_j$$

where $X_i, Y_i, Z_i (1 \leq i \leq n)$ are binary 0-1 variables, and for $1 \leq i \leq n$,

[4]We would like to thank Professor Egan Balas of the Graduate School of Industrial Administration at Carnegie Mellon University for suggesting the linearization technique for the case of $M = 2$ upon which this is based.

$f_i \geq 0$
$f_i - Fi \leq 0$
$f_i - X_i \leq 0$
$-f_i + F_i + X_i \leq 1$

$g_i \geq 0$
$g_i - Fi \leq 0$
$g_i - Y_i \leq 0$
$-g_i + F_i + Y_i \leq 1$

$h_i \geq 0$
$h_i - Fi \leq 0$
$h_i - Z_i \leq 0$
$-h_i + F_i + Z_i \leq 1$

$Z_i - X_i \leq 0$
$Z_i - Y_i \leq 0$
$X_i + Y_i - Z_i \leq 1$

MRSQP can be efficiently solved using standard optimization techniques for mixed integer programming. This is particularly so for problems where the number of applications is relatively small (a few tens at most). Commercial software packages that are available for these purposes include *lindo* [14], *gino* [13], *C-Whiz* [5] and *CPlex* [4]. We provide a brief evaluation of how efficiently this linear mixed integer programming problem can be solved in Section 3.7.

3.5. Dealing with $M \neq 4$

The definition of MRSQP in Section 3.4 assumed that $M = 4$ schemes were available to support and implement a particular QoS dimension. The problem scales readily if $M = 2^p$, where p is a positive integer. The binary variables X_i and Y_i are then replaced by the binary variables $X_{il}, 1 \leq i \leq n$ and $1 \leq l \leq p$ and the linearization process is repeated after the objective function is expanded accordingly.

If M is *not* a power of 2, pick the smallest integer p such that $2^p > M$ and define the objective function and constraints assuming that 2^p schemes are available (as described above). Then, set $u_{ik} = -1$ and $a^j_{ik} = 0, (M+1) \leq k \leq 2^p, 1 \leq j \leq m$. The optimization will be forced away from choosing the non-existent schemes.[5]

3.6. Dealing with The Same Utilities for All M Schemes

Suppose that all alternative schemes available for implementing the QoS dimension provide the same maximal utility to each application. For example, consider a bulky real-time data stream being transmitted and the end-to-end delay of the data stream is the QoS dimension under consideration. In addition, the data must be received at its destination without any loss. There can be several options available. The data can be transmitted as is, or it can be compressed using any one of several available lossless compression schemes (such as "zip" with different compression factors). Any scheme chosen impacts the demand on processing and network resources differently. However, since there is no data loss in the compression, at the receiver site, the utility derived from each scheme is the same (for a given end-to-end delay).

Under such conditions, where $u_{i1} = u_{i2} = u_{i3} = u_{i4} = u_i$, the objective function of MRSQP can be simplified as follows:

Objective:

$$\text{maximize} \sum_{i=1}^{n} u_i F_i$$

subject to $0 \leq F_i \leq 1$ and the exact same constraints as MRSQP.

The outlines of a heuristic that is expected to give practically very acceptable but suboptimal results at much higher speeds than the mixed integer programming solution defined above is presented in [21]. As a result, it may be used even within a non-preemptive kernel, for example.

3.7. Solving MRSQP in Practice

As shown in Section 3.4, MRSQP is expressed as a linearized mixed integer programming problem. However, it must be noted that integer programming problems can still potentially have long execution times. In order to obtain some insights into its run-time efficiency, we solved these problems in a commercially available optimization package called CPLEX [4] for some sample situations. We used a 266MHz Pentium with 64MB of memory for performing the optimization.

In MRSQP, the number of constraints in the problem is 62 when 10 applications are considered, and 122 when 20 applications are considered simultaneously. The results are summarized in Table 1.

The first two columns of Table 1 represent inputs to MRSQP, defining the objective function, the resource constraints and the number of schemes (M) available to support the specified QoS. The number of iterations exercised by the optimizer, the time spent in pre-solving the problem (where redundant constraints are eliminated and pseudo-cost variables are selected) and the time to solve the problem are listed in the last three columns respectively.

We drew some of the following conclusions based on our observations:

- The solution times of about 20 *ms* is a strong indicator that this mixed integer programming problem can be solved in real-time for online decision making of resource allocation. We found that in most cases, the solution is found in about 0.02 seconds, but we suspect that the running time is often much less since the resolution of the clock itself is 0.01 seconds.
- The solution time in the small 10s of milliseconds also indicates that such a decision should *not* be computed as a

[5] Alternatively, one can also assign $\forall i, 1 \leq i \leq n, b^j_{ik} = \infty, (M+1) \leq k \leq 2^p$.

# of Applications	M	# of Iterations	Pre-solve Time (s)	Solution Time (s)
10	2	41	0.00	0.01
10	2	43	0.01	0.02
10	2	29	0.01	0.03
10	2	30	0.00	0.01
10	2	42	0.00	0.01
20	2	63	0.01	0.03
20	2	68	0.01	0.01
20	2	53	0.01	0.02
20	2	61	0.01	0.02

Table 1. The Run-Time Efficiency of Solving MRSQP.

long critical section in a non-preemptive kernel. Instead, this decision should be made by a resource manager running preemptively in user space. This is the architecture of the QoS manager used in RT-Mach [15].

- Another interesting observation from these and many other similar experiments was that all applications except for one application have their $F_i = 0$ or 1. Under relatively rare conditions, two applications have their $F_i \neq 0$ or 1. We have yet to see a case where three applications have their $F_i \neq 0$ or 1. We expect that this observation can form the basis of an efficient heuristic that can be used to solve substantially larger problems in real-time.

4. Concluding Remarks

In this paper, we have presented practical solutions to some important problems that were not addressed by the QoS-based Resource Allocation Model of [20]. For example, this prior work presented only a greedy algorithm to apportion an available resource across multiple applications to satisfy multiple QoS dimensions (at least one of which is dependent on another). We have shown in this paper that the problem of finding the optimal resource allocation is NP-hard and present a brute-force scheme to find this allocation. However, a simple polynomial algorithm based on computational geometry allowed us to find a solution within a provably fixed and very short distance from the optimal allocation. We have also studied the problem of apportioning multiple resources to satisfy a single QoS dimension, the converse of a problem solved in [20]. This problem is rather interesting because resources can be traded against each other while satisfying the same QoS requirement. We solve this problem as a linear mixed integer programming problem that is solvable efficiently using commercially available software packages and libraries to obtain an optimal resource allocation.

We expect that a deep understanding of these and related solutions will yield insights into the general problem of apportioning multiple resources to satisfy simultaneously multiple QoS dimensions of multiple concurrent applications. In addition, extending some of these solutions to higher dimensions remains to be carried out. We are also exploring heuristics that can be used to solve MRSQP at even higher efficiencies (for use inside a real-time kernel for example). Finally, a significant amount of empirical work must be done to determine resource consumption functions.

Acknowledgments

The authors would like to thank the other members of the Amaranth project at Carnegie Mellon University.

References

[1] 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.

[2] J. C. R. Bennett, D. C. Stephens and H. Zhang. *High Speed, Scalable, and Accurate Implementation of Fair Queueing Algorithms in ATM Networks.* ICNP '97.

[3] J. C. R. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675-689, Oct 1997. Also in Proceedings of SIGCOMM'96, Aug, 1997.

[4] Using the CPLEX Callable Library. Using the CPLEX Base System with CPLEX Barrier and Mixed Integer Solver Options. CPLEX Division, ILOG Inc., 1997.

[5] C-Whiz Manual. *C-Whiz Linear Programming Optimizer.* Ketron Management Science, July 1994.

[6] 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.

[7] T. Cormen, C. Leiserson, R. Rivest *Introduction to Algorithms*. The MIT Press, 1990. ISBN 0-262-03141-8 (MIT Press), ISBN 0-07-013143-0 (McGraw-Hill)

[8] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches*. MIT Press, 1988. ISBN 0-262-09027-9.

[9] 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.

[10] 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.

[11] 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.

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

[13] J. Liebman, L. Lasdon, L. Schrage and A. Waren. *Modeling and Optimization with GINO*. The Scientific Press, 1988.

[14] Lindo Manual. *Optimization Modeling with LINDO*. 5th Edition. Duxbury Press, 1997.

[15] 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.

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

[17] 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.

[18] R. Neopolitan, and K. Naimipour. *Foundations of Algorithms*. 2nd Edition, Jones and Bartlett, 1998. ISBN 0-7637-0620-5.

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

[20] 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*. December, 1998.

[21] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Some Solutions for QoS-based Resource Allocation In *Technical Report, Department of Computer Science, Carnegie Mellon University*. June, 1998.

[22] R. Sedgewick. *Algorithms in C*. Addison Wesley, 1990. ISBN 0-201-51425-7.

[23] 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.

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