Performance Analysis of Dynamic Soft Real-Time Systems

Scott A. Brandt
Computer Science Department
University of California, Santa Cruz
sbrandt@cse.ucsc.edu

Abstract

Soft real-time processing is real-time processing in which some or all applications are allowed to miss deadlines, particularly in situations of system overload. Resource allocation decisions in such systems are often based in part on specifications of application utility or benefit and directly affect which applications will miss deadlines and by how much. A feedback mechanism is often provided to inform the applications of their current resource allocations, allowing them to modify their processing to provide the best possible performance. These characteristics result in a situation in which traditional measures of performance, both general-purpose and real-time, can provide incomplete or inaccurate measures of system and application performance. In this paper we examine these issues and shortcomings, identify performance factors that need to be measured to address them, and propose specific metrics to quantify these performance factors. We conclude with results from the application of these metrics to our QoS Level soft real-time system.

1. Introduction

Many researchers have investigated mechanisms for supporting soft real-time processing [3,7,11,12,19,20,25, 26]. The basic assumptions in these systems are that some or all of the applications can miss deadlines and still perform satisfactorily, and that situations can occur in which the worst-case resource needs of all of the running applications may not simultaneously be satisfiable. To support this type of processing, these systems variously provide a static resource guarantee with the possibility of renegotiation of those guarantees [13,20], or dynamic assignment of resources based on a specified application importance, utility or benefit (hereinafter called "benefit") for each application [4,7,11,17,20,24,25]. Those systems that provide a static resource guarantee must also have some notion of the relative benefit of the applications if they are to have any meaningful renegotiation of the guarantees when the resources become oversubscribed [14]. In all of these systems, the benefit specification can be thought of as a function of an application's benefit versus the resources it receives (in those systems with a single number *b* representing application benefit, it can be thought of as a linear function of the resources with value ranging from 0 to *b*). Given these specifications, a resource allocation mechanism reads all applications' benefit functions and parcels out the resources in an attempt to optimize the allocation according to various goals, usually fairness or overall benefit.

In all of these soft real-time systems it is the responsibility of the applications to run correctly within the less-than-optimal resource allocations that they may receive. Some systems provide direct support for the applications to do this [1,2,17,24,25], while others provide information to the applications to allow them to do this, but without providing any direct support [7,13,19,20]. In all cases, the applications may simply run slower, which is the result of running the same algorithm with less resources. However, many of these systems support and expect that applications will choose to actively modify their resource usage in other ways based on the resources that they have been granted [4,13,17,19,20,24,25].

An important question that arises in all of these systems is how to measure soft real-time system and application performance. Traditional general purpose and real-time performance metrics provide important information about how soft real-time applications and systems perform, but they may fail to fully characterize the soft real-time aspects of the performance. General-purpose metrics do not characterize the real-time nature of the processing because they do not address deadlines or time-based degradation of an application's output. Traditional real-time metrics such as latency and jitter can be used to characterize the time-based nature of the processing in many important ways, but because they have no notion of missed deadlines, benefit, and time-based degradation of output

quality (i.e. how quality varies as deadlines are missed by greater or lesser amounts), they also miss important aspects of soft real-time processing.

In systems where applications miss deadlines without changing any other aspect of their processing, the performance of the applications can be characterized directly by measuring the deadline misses. The number and frequency of missed deadlines and the amount by which they are missed are all important characterizations of this aspect of soft real-time performance. However, these measures may fail to fully characterize the impact of the deadline misses. Specifically, without looking at the benefit specifications, the characterization of the relative importance of the application and the benefit provided at various levels of performance, it is impossible to quantify the impact of the deadline misses. For those applications with non-linear benefit functions this effect may be amplified inasmuch as they may provide significantly different amounts of benefit for small changes in allocated resources, deadlines missed, or amount by which deadlines are missed. This benefit information, and the output quality and user satisfaction that it reflects, is absent in any simple measure of deadline misses.

In systems with allocation-dependent application processing, i.e. systems where the applications are informed of the resources that they are receiving and which can change their processing accordingly (rather than just missing deadlines), traditional metrics, and any measure of missed deadlines, can in fact fail to characterize the performance at all. Consider a 24-bit color multimedia video application that is running at 24 frames per second without missing any deadlines. Upon learning that it will receive less resources this application could decide to continue to run at 24 frames per second but switch to monochrome images. The resulting execution could easily have the same latency, jitter, and frame rate, and continue to run without missing deadlines, while using less system resources. Any metric that ignores the benefit of the application will fail to capture the important differences between these two modes of operation. And because the change in benefit provided by the application may be a non-linear function of the resources consumed, even a measure of resources consumed by the application is at best an imperfect characterization of the processing. We conclude that the application performance is best characterized by measuring the benefit provided by the application as it executes.

To address these issues, we present two soft real-time metrics. We look at time-varying behavior of the system in terms of how it manages the resources allocated to the applications by examining the benefit that results from those allocations. Maximizing benefit is the explicit goal of most resource managers in these systems, and so a direct measurement of the benefit provided by the applications gives a good indication of how the system is performing. Accordingly, we use a measurement of the benefit provided by the running applications as one of our soft real-time performance metrics. This measured benefit is a direct characterization of the performance of the system, including the performance of the resource allocation algorithm and any overhead consumed by the system and the resource manager. Given equivalent applications and benefit specifications, it can further be used for direct comparison of different resource allocation algorithms and soft real-time systems.

Because applications can and will enter and leave the system, individual applications' resource needs and benefit functions may change during the running time of the applications, and relative importances of the applications may also change, the amount of resources granted to the applications will change over time. This change in allocated resources is directly reflected in a change in the benefit provided by the applications. This possibly frequent change in resource allocations and resulting benefit suggests that efficient resource allocation algorithms are needed. The change in measured benefit as the applications execute is not necessarily a problem for the applications themselves. The change will represent various changes in the application performance, hopefully optimized to minimize the impact on the user. However, for some applications, notably interactive applications, change in benefit, directly reflecting changes in the operation of the applications, can itself have a non-trivial cost to the user. For example, running a video application at two different sizes may yield two different but acceptable amounts of benefit. However, rapidly alternating between the two different sizes could easily result in a perceived benefit to the user that is worse than either of the two benefits resulting from stable operation at either of the two sizes. We measure this lack of stability as the frequency and distance of the change in benefit over time. Intuitively, benefit focuses on *maximizing* the benefit over time, and stability on minimizing the *frequency* and *amplitude* of the benefit changes; in some cases benefit reflects the desired soft real-time characteristics, and in others it is stability.

In addition to characterizing the performance of soft real-time applications and systems, these metrics allow one to specify a meaningful optimal allocation strategy (beyond simply maximizing benefit), to measure the performance of a given resource allocation algorithm, and to compare the performance of the algorithms to other algorithms and to a calculated optimal solution. They also allow for the direct comparison of soft real-time systems. Finally, they allow for a useful classification of resource allocation algorithms according to the kind of service they

provide, and applications according to the kind of service they require.

We conclude the paper with results from calculating these metrics on our QoS Level soft real-time system [3,4,8,22,23] with a variety of representative resource allocation algorithms with several different sets of applications. The results show that even the simplest distributed algorithm performed within 15% of our calculated optimal solution with respect to benefit. The studies show a much larger variation with respect to stability. Interestingly, our simplest centralized algorithm performed the best with respect to both benefit and stability.

2. Background

One of the key assumptions in soft real-time is that the worst-case resource requirements of all running applications are not guaranteed to be met. When the applications begin to miss deadlines they may continue without corrective action [12,19], the system may attempt to shed load through preemption [5], or the applications may cooperatively adjust their processing to deal with a less-than-optimal resource allocation [3,20,24]. In this latter class of systems, the decision of how to allocate the resources among the applications and how much resources each application will receive is not trivial. In an attempt to manage the resources so as to maximize user satisfaction, many researchers have employed the idea of a user specification of application benefit as an indication of the relative importance of the running applications. The benefit of the individual applications is generally in the form of a function of application benefit vs. resources received. The function can be in the form of a constant (equivalent to a single number), a discrete step function, or a continuous function. Resource allocation schemes in such systems attempt to provide optimal service to the user with respect to some goals such as fair allocation to the applications, a guaranteed minimum level of service, or maximum overall system benefit. Applications receiving less than their required resource allocation can either miss deadlines (equivalent to lengthening their period if they miss the deadlines by some constant amount), modify their processing so as to run acceptably within the resources they have been allocated, or some combination of the two.

Researchers in the area have developed a variety of different solutions. Some systems have provided processor reservation mechanisms that allows an application to request a certain guaranteed amount of resources [13,19]. The resource allocation mechanism is extremely simple - if the request can be granted with the currently available resources, it is. SMART used a fair-share resource allocation mechanism that gave each application a percentage of the overall resources proportional to its relative importance [20]. In our work with QoS Level soft real-time, we have explored a variety of resource allocation algorithms that are variously intended to approximate an optimal solution, run with as little overhead as possible, provide stable allocations, etc. Researchers at CMU have presented algorithms for some specific resource allocation problems that arise in quality of service soft real-time [17,24].

3. Benefit-Based Soft Real-Time Performance Metrics

Because benefit, in one form or another, is used as an input to most soft real-time systems to characterize the quality of the output of each application relative to the amount of resources it receives, and because maximizing benefit is the implicit or explicit goal of most of these systems, we feel that it should also be used in part to characterize system and application performance. Consequently, we evaluate the behavior of the resource allocation and the system as a whole in terms of how they manage the benefit of the applications. From the application perspective, a resource allocation manager will provide varying amounts of resources. As a result, the behavior of the system can be represented by metrics to represent the way the benefit change as each application executes. *Benefit* therefore refers to the amount of benefit provided by the applications as they execute, *stability* refers to the frequency and amount of benefit changes during the execution. These two metrics apply equally well to any system that employs any kind of value functions to characterize soft real-time.

Benefit, which characterizes the quality of the output of a process, is defined as follows. If $b_i(r)$ is the benefit function for application *i* as a function of the amount of allocated resources and $r_i(t)$ is the actual resource allocation provided to application *i* at time *t*, then we can define $B_i(t) = b_i(r_i(t))$ to be the actual benefit provided by application *i* at time *t*, or the *output benefit* of application *i*. Given the output benefit of all applications over a span of time, we define our specific benefit metrics as follows:

At any particularly time *t*, $B_i(t)$ is the *instantaneous benefit* of application i. During any interval of time (t_1, t_2), the *total benefit* is defined as $B_i(t_1, t_2) = \int_{t_1}^{t_2} B_i(t)dt$. With functions on discrete values, the benefit curve from t_1 to t_2 is a step function with *m* distinct intervals each lasting time d_j, and the integral is a discrete sum,

$B_i(t_1, t_2) = \sum_{j=1}^{m} B_i(t_1 + d_j - \varepsilon) \times d_j$, where $\varepsilon < \min(d_j)$. The *average benefit* $\bar{B}_i(t_1, t_2)$ from time t_1 to t_2 is simply $\left(\frac{1}{t_2 - t_1}\right) B_i(t_1, t_2)$. The instantaneous, total, and average benefit for a set of running applications ($B(t)$, $B(t_1, t_2)$, and $\bar{B}(t_1, t_2)$)are simply the sum of the individual application benefits.

Benefit gives a good indication of the quality of the output of a process, but it is incomplete. In particular, moving from one output benefit to another may indirectly affect the quality of an application. For example, a video player that continuously runs at 15 frames per second is far preferable to one that alternates between 0 and 30 frames per second every few seconds, even though their average benefit will be the same. To measure this variation between levels we define the *instability* of each application as follows.

At any time *t*, the *instantaneous instability* of process *i* is the slope of the curve of the output benefit, thus $I_i(t) = \left|B_i'(t)\right|$. For applications defined on discrete resource amounts, the instantaneous instability is infinite during the transition from one level to another and 0 at all other times. During any interval of time (t_1, t_2), the *total instability* is defined as $I_i(t_1, t_2) = \int_{t_1}^{t_2} I_i(t)dt$. As above, for applications defined on discrete resource amounts the benefit curve from t_1 to t_2 is a step function with *m* distinct intervals each lasting time d_j, and the integral is a discrete sum, $I_i(t_1, t_2) = \sum_{j=1}^{m} \left|(B_i(t_1 + d_j + \varepsilon) - B_i(t_1 + d_j - \varepsilon))\right|$, where $\varepsilon < \min(d_j)$. Conceptually, the instability of an application from time t_1 to t_2 is simply the amount of benefit change during that time. The *average instability* $\bar{I}(t_1, t_2)$ of application *i* from time t_1 to t_2 is $\left(\frac{1}{t_2 - t_1}\right) I_i(t_1, t_2)$. Instantaneous, total, and average instability for a set of running applications ($I(t)$, $I(t_1, t_2)$, and $\bar{I}(t_1, t_2)$) are defined as the sum of the values for the individual applications.

The benefit and instability metrics provide a concrete measure of soft real-time, and they allow one to specify a meaningful optimal allocation strategy, to measure the performance of a given resource allocation algorithm with a set of applications, and to compare the performance of the algorithms to other algorithms and to an offline calculated optimal solution. These metrics also allow one to quantify the performance of the individual algorithms so that they can be compared across soft real-time platforms and with a variety of application suites.

One result of the analysis using these metrics is the observation that different resource allocation algorithms may provide different kinds of performance. Intuitively, some algorithms may provide higher benefit at the cost of instability, others may provide higher stability at the cost of average benefit, and still others may provide performance somewhere in between. Furthermore, for some applications, benefit is most important while stability may not really matter (e.g., in a real-time image processing system). In other cases, e.g., an application that changes the effective graphic display frame rate, stability is an important factor since maximized benefit with an instable frame rate is highly undesirable at the user interface. Classification of the algorithms according to the metrics is an important result of our work and provides evidence for the correctness of the metrics themselves.

These metrics neatly characterize the performance of the running applications. Number and frequency of missed deadlines are included directly in the metrics by considering a missed deadline to be an application period with benefit equal to 0. The metrics can be used to compare two applications to each other and to compare applications with a computed (off-line) optimal solutions (when complete and reliable resource usage information is available). Instantaneous values and averages over a short time can be computed to determine worst-case performance information and long-term averages can be used to get an idea of general performance. Averages and standard deviations over multiple executions of the same application suite can give an indication of the repeatability of the results.

Another item of interest is system overhead. In comparing two algorithms or systems, the effect of system overhead is automatically captured by the resulting benefit and instability measurements. If the system is using a significant amount of CPU time, the cycles are not available to the applications and their resulting lower level of execution will be reflected in the benefit metric.

4. Our Soft Real-Time System

Our soft real-time system [3,4,8] is based on the notion of *Quality of Service* (QoS) *levels* [25]. The soft real-time resource and timing requirements of an application are abstracted into a table relating the value of a computation to a level of resource allocation. In essence, the application designer evaluates the soft real-time criteria that are important for that application, writes the applica-

tion according to those criteria, then derives a table with multiple levels to reflect the quality of the result that can computed as a function of the amount of resources allocated to it. The highest level represents a target resource allocation amount with a benefit that will result if the system can allocate the corresponding amount of resources; lower levels reflect lower benefit for the reduced resource allocation. Once each application's QoS levels have been specified, the resource manager uses the benefit-allocation information to allocate resources according to the amounts needed versus the global benefit that can be achieved across the applications. Because our system is implemented as middleware, it employs a cooperative application model in which applications are expected to run at the system-specified level.

5. Results

This section presents the results of running our system with various application suites. All of the experiments were executed on a 200 Mhz Pentium Pro system running Linux 2.0.30. All applications and middleware were executed using the standard best-effort Linux scheduler. In order to generate data, multiple copies of two representative applications were used in various combinations. The applications are two mpeg players employing different soft real-time strategies. The first, mpeg_rate, modifies its frame rate in order to change its resource usage. Mpeg_rate has 6 levels and varies its resource usage from 48.9% of the CPU down to 9.7%. The second application, mpeg_size, modifies its display size. mpeg_size has 8 levels and varies its resource usage from 86.4% of the CPU down to 33.0%. Combinations and multiples of these applications with varying start and end times and different overall and relative benefit values were executed and measured under the algorithms described below.

Figure 1 shows a representative set of benefit curves from executing the system with one copy of each of the two applications using the Proportional algorithm (see below). For this graph, Benefit is sampled every 1/10 of a second for 60 seconds. Application 1 starts at sample 10 and immediately goes to its highest level. Application 2 starts at sample 60 and causes application 1 to miss some deadlines. The system changes the levels of the applications in response to the missed deadlines and the applications stabilize by about sample 90 with 1 additional missed deadline at about sample 170. Application 1 terminates at sample 510 and application 2 terminates at sample 560. For this experiment, $B(t_1,t_2)$ is 493.90 and $I(t_1,t_2)$ is 84.52. There were 6 missed deadlines out of 880 total application periods.

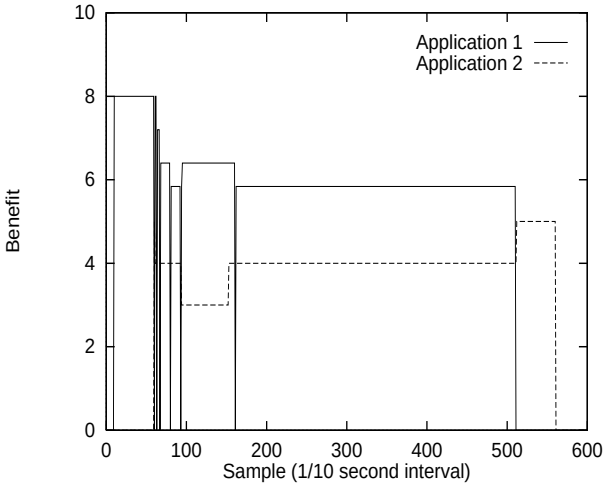

Figure 1: Benefit Provided by Two Applications

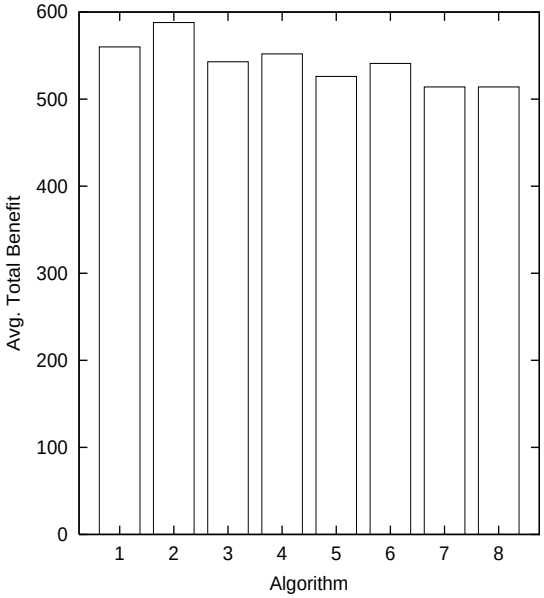

Figure 2: Average Benefit for All Algorithms

In order to examine the metrics we ran 3 different application scenarios 20 times with several different resource allocation algorithms (discussed below) and measured average values and standard deviations for Benefit and Instability. The application scenarios had 2, 3, and 4 applications with varying absolute and relative benefit values and start and end times. In 20 executions of a single scenario typical standard deviations were less than 2% with none greater than 4%. Figure 2 shows a summary of the average Benefit for the entire set of experiments and Figure 3 shows a summary of the average Instability for the entire set of experiments. Each set of experiment

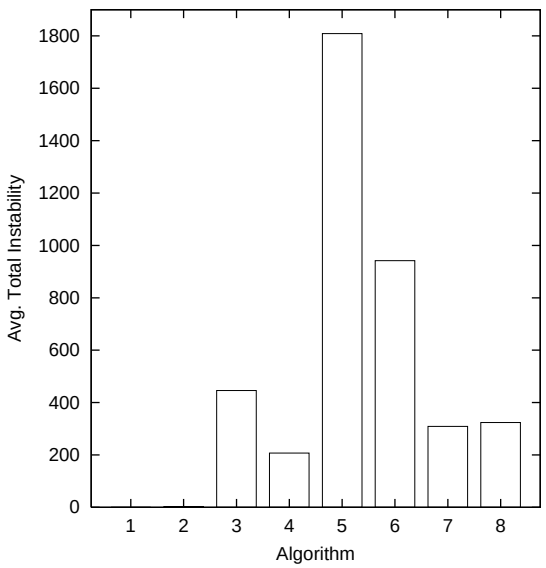

Figure 3: Average Instability for All Algorithms

showed roughly the same relative values and the averaged values serve to show the trends with respect to the metrics.

The algorithms employed in these experiments were not necessarily expected to provide optimal performance but to either approximate those written about in the literature or perform reasonably well with respect to one or both metrics. The algorithms examined were:

1. Calculated Benefit Optimal — A simulation that calculates the highest obtainable benefit (assuming accurate resource usage estimates). The benefit for this simulation serves as a useful benchmark for analyzing the benefit performance of the other algorithms.

2. Calculated Stability Optimal — A simulation that calculates the highest obtainable benefit with no instability - applications start at a predetermined level and remain at that level for the duration of their execution. This algorithm shows the best performance that can be expected assuming *a priori* knowledge of the applications and a requirement of zero instability, analogous to hard real time performance. In the average, none of the algorithms examined performed as well as this simulated algorithm. However, Proportional provided somewhat higher benefit in the four application experiment.

3. Distributed — A decentralized algorithm in which each application lowers its level whenever it misses a specified number of deadlines and raises its level whenever it has run for a certain number of periods without missing a deadline. This algorithm is similar to that presented by Tokuda [25]. This extremely simply algorithm did surprisingly well in terms of Benefit, with moderate instability relative to the other algorithms.

4. Proportional — A centralized algorithm in which each application is allocated a percentage of the available resources proportional to its benefit. This algorithm is based loosely on the resource allocation algorithm employed in SMART [20,21]. This algorithm performed the best with respect to both metrics.

5. Optimal Benefit — A centralized algorithm that employs a brute-force optimization to determine the resource allocation that maximizes overall benefit. This algorithm should theoretically produce the highest obtainable benefit. However, the overhead of running the algorithm results in a significant number of deadline misses and correspondingly high instability. Because it is frequently changing the resource allocations in response to the missed deadlines, the benefit is correspondingly lowered as well.

6. Hybrid — A centralized algorithm that begins with the optimal allocation, then employs a limited depth search to determine the resource allocation within one level per application of the current allocation that maximizes overall benefit. This algorithm should have less overhead than optimal and less instability, but may also result in suboptimal benefit. The results show that the instability is about 50% of that of Optimal, a result of the reduced distance of the level changes allowed as well as the reduced overhead. Because of the reduced instability, the benefit is also somewhat higher.

7. Greedy (Benefit) — A centralized algorithm that begins with the optimal allocation, then uses a greedy approach to determine subsequent allocations as necessary due to variations in resource availability, missed deadlines, etc. This algorithm greedily chooses to modify the level of a single application by maximizing the amount of benefit gained and minimizing the amount of benefit lost with each change. This algorithm is similar to that employed by Abdelzaher [1]. This algorithm should have less overhead than the optimal algorithm but was expected to yield near-optimal performance as it makes level changes near the optimal solution. While the instability is moderate, the benefit is the lowest of the algorithms examined. Closer examination reveals the reason for this: because applications enter and exit the system at different times, there is no single point at which the optimal solution can be calculated. Consequently, this greedy algorithms never starts at an optimal point and apparently never gets close to one.

8. Greedy (Benefit Density) — A second greedy algorithm that chooses to modify the level of a single application by maximizing the benefit density gained and minimizing the benefit density lost with each change, where benefit density is benefit divided by the CPU usage. This algorithm was inspired by an algorithm proposed by Jensen in the context of deadline miss soft real-time scheduling algorithms [11]. It was hoped that this algorithm would provide better stability than Greedy Benefit while still maximizing benefit. The results show that this algo-

rithm did no better than Greedy Benefit in terms of Benefit and did slightly worse in terms of Instability.

The results presented are interesting for several reasons. Primarily, they show that the metrics are useful in analyzing and comparing different resource allocation algorithms. Among the algorithms we examined, there was relatively little variation in the Benefit provided, ranging from 87% to 94% of the maximum obtainable benefit. Within this small range, the simplest centralized algorithm, Proportional, outperformed all of the rest and Distributed, a trivial decentralized algorithm, was second best. There was significantly more variation in terms of Instability with almost an order of magnitude difference between the best and the worst performers. In this metric, Proportional was best, with the two greedy algorithms tied for second place. Surprisingly, Distributed also had relatively low instability.

6. Conclusion

With respect to soft real-time applications, the most important question is what kind of algorithm yields the best application performance. As we have discussed, the best performance in general comes from maximum benefit and no instability, but this is not generally achievable in practice. With benefit functions on discrete resource amounts the problem is NP-Complete, being trivially reducable to the Knapsack Problem [6]. Furthermore, any approximation that maximizes benefit is likely to result in large variations in individual allocations with small variations in available resources, and consequently have large instability. While continuous benefit functions may not exhibit this particular behavior, they impose much greater difficulty on the programmer inasmuch as they require that the applications be able to deal with any resource allocation that they may be granted. With some resources (such as CPU) this is perhaps a reasonable expectation, but with others (such as memory) it is not. In general it is expected that for some resources it will not be possible to specify a continuous benefit function.

Clearly high benefit and low instability are both desirable, but what is less obvious is that these two conditions are sometimes mutually exclusive. In the extreme case it is clearly not possible to optimize for both benefit and instability since a perfectly stable system will never adjust the allocations to applications resulting in a suboptimal use of the resources. In the situation where resource availability is fluctuating, low instability requires that application levels remain relatively constant but high benefit requires that application levels change to take advantage of the available resource. Such fluctuations can be caused by many factors including applications entering or exiting the system, application resource usage changes (as with application mode changes), changes in user-assigned benefit numbers, execution of sporadic tasks, inaccurate resource usage estimates, *etc*. Conservative estimates of resource usage and conservative algorithms will result in relatively few level changes, yielding relatively low benefit numbers but correspondingly low instability. By contrast, aggressive resource usage estimates and aggressive algorithms will result in high benefit numbers with correspondingly high instability. No algorithm will be able to simultaneously optimize both metrics. The conclusion is that there is no single optimal algorithm. An algorithm can optimize for benefit by responding immediately to changes in resource availability, or it can optimize for stability by choosing a level and executing at that level forever regardless of changes in resource availability, or it can compromise and provide performance that is adequate but imperfect in both axes.

Accordingly, we see that we can classify applications according to the kind of algorithm performance that results in good application performance. In essence we have a spectrum of possible algorithm performance with hard real-time (highest benefit achievable with zero instability) at one end and best effort (highest benefit achievable without regard to instability) at the other. Various types of soft real-time (e.g. stability-centric or benefit-centric) fall in different places along this spectrum. It should be clear that all algorithms will not satisfy all applications. Different applications will require different processing characteristics and will therefore best be supported by different algorithms. In those cases where applications with different preferences are running simultaneously, an algorithmic choice will have to be made, perhaps by choosing a compromise or by choosing the algorithm most appropriate for whichever application is most important. A natural question that arises is whether or not the metrics actually serve to distinguish between the different resource allocation algorithms that have been proposed or used in soft real-time systems. Results presented in Section 5 demonstrate that they do.

The results of our experiments with the metrics show that they provide meaningful information about the algorithms. We expected to conclude that no single algorithm would simultaneously provide the best performance in terms of both metrics, but our results suggest that the simple Proportional algorithm is the best choice for most applications. However, more research with a wider range of application sets needs to be done before this can be concluded in general. In particular, it seems obvious that a simple admission-based policy that assigns appropriate levels at the time of admission should be able to provide better performance with respect to Instability.

References

[1] T. Abdelzaher and K. Shin, "End-host Architecture for QoS-Adaptive Communication", *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium*, Jun. 1998.

[2] S. Brandt, "Soft Real-Time Processing with Dynamic QoS Level Resource Management", Ph.D. Thesis, University of Colorado, June, 1999.

[3] S. Brandt, G. Nutt, T. Berk, and M. Humphrey, "Soft Real-Time Application Execution with Dynamic Quality of Service Assurance", *Proceedings of the 6th IEEE/IFIP International Workshop on Quality of Service*, pp. 154-163, May 1998.

[4] S. Brandt, G. Nutt, T. Berk, and J. Mankovich, "A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage", *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 307-317, Dec. 1998.

[5] C. Compton and D. Tennenhouse, "Collaborative Load Shedding", *Proceedings of the Workshop on the Role of Real-Time in Multimedia/Interactive Computing Systems*, Nov. 1993.

[6] T. Cormen, C. Leiserson, and R. Rivest, Introduction to Algorithms, The MIT Press, 1990.

[7] C. Fan, "Realizing a Soft Real-Time Framework for Supporting Distributed Multimedia Applications", *Proceedings of the 5th IEEE Workshop on the Future Trends of Distributed Computing Systems*, pp. 128-134, Aug. 1995.

[8] M. Humphrey, T. Berk, S. Brandt, G. Nutt, "The DQM Architecture: Middleware for Application-centered QoS Resource Management", *Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, pp. 97-104, Dec. 1997.

[9] T. Ibaraki and N. Katoh, *Resource Allocation Problems, Algorithmic Approaches*, The MIT Press, Cambridge, Massachusetts, 1988.

[10] O. Ibarra and C. Kim, "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems", *Journal of the ACM*, 22(4):463-468, October 1975.

[11] E. Jensen, C. Locke and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proceedings of the 6th IEEE Real-Time Systems Symposium*, pp. 112-122, Dec. 1985.

[12] M. Jones, J. Barbera III, and A. Forin, "An Overview of the Rialto Real-Time Architecture", *Proceedings of the 7th ACM SIGOPS European Workshop*, pp. 249-256, Sep. 1996.

[13] M. Jones, D. Rosu, M. Rosu, "CPU Reservations & Time Constraints: Efficient Predictable Scheduling of Independent Activities", *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.

[14] K. Kawachiya, M. Ogata, N. Nishio and H. Tokuda' "Evaluation of QoS-Control Servers on Real-Time Mach", *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 123-126, Apr. 1995.

[15] J. Kay and P. Lauder, "A Fair Share Scheduler", *Communications of the ACM*, 31(1):44-55, Jan. 1988.

[16] E. Lawler, "Fast Approximation Algorithms for Knapsack Problems", Mathematics of Operations Research 4(4): 339-356, 1979.

[17] C. Lee, R. Rajkumar and C. Mercer, "Experience with Processor reservation and Dynamic QoS in Real-Time Mach", *Proceedings of Multimedia Japan*, Mar. 1996.

[18] H. Massalin, and C. Pu, "Fine-Grain Adaptive Scheduling using Feedback", *Computing Systems*, 3(1):139-173, Winter 1990.

[19] C. Mercer, S. Savage and H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications", *Proceedings of the International Conference on Multimedia Computing and Systems*, pp. 90-99, May 1994.

[20] J. Nieh and M. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications", *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.

[21] J. Nieh and M. Lam, "Integrated Processor Scheduling for Multimedia", *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Apr. 1995.

[22] G. Nutt, T. Berk, S. Brandt, M. Humphrey, and S. Siewert, "Resource Management of a Virtual Planning Room", *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, Sep. 1997.

[23] G. Nutt, S. Brandt, A. Griff, S. Siewert, T. Berk, and M. Humphrey, "Dynamically Negotiated Resource Management for Data Intensive Application Suites", *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[24] R. Rajkumar, C. Lee, J. Lehoczky and D. Siewiorek, "A Resource Allocation Model for QoS Management", *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[25] H. Tokuda and T. Kitayama, "Dynamic QoS Control based on Real-Time Threads", *Proceedings of the 3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Nov. 1993.

[26] L. Welch, B. Ravindran, B.A. Shirazi, and C. Bruggeman, "Specification and Modeling of Dynamic, Distributed, Real-Time Systems", *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 72-81, Dec. 1998.