Performance Metrics for Soft Real-Time Systems

Scott Brandt
Gary Nutt

CU-CS-876-98

Performance Metrics for Soft Real-Time Systems

Scott Brandt and Gary Nutt
University of Colorado at Boulder
Department of Computer Science
{sbrandt,nutt}@cs.colorado.edu

Abstract

Soft real-time applications are becoming increasingly common in today's systems, primarily because of the use of continuous media. These applications are characterized by their need to have parts of their computations accomplished by prespecified deadlines. There is an emerging community of soft real-time researchers that use quality of service (QoS) levels as a means for characterizing soft real-time requirements, and who are designing systems based on QoS levels. Despite the use of QoS levels, there is no accepted set of metrics by which one can compare and evaluate different systems according to general soft real-time criteria. Metrics are important to application designers, since without them a designer has no means for tuning applications and/or comparing system support performance. We have derived two metrics, based on QoS levels, for comparing soft real-time system behavior: Benefit and instability. Benefit reflects the time-resource product of resource allocation, and instability reflects the rate at which resource allocation levels can change during an application repertoire's execution. In this paper benefit and instability are formally defined, then we use them to compare the performance of various soft real-time resource allocation policies.

While one would expect a policy to excel either on the basis of benefit or on instability, applying the metrics to several published policies showed that one of them produced the best result on both.

1 Introduction

Systems that support soft real-time applications are becoming increasingly important in mainstream computing because of the desire to makes heavy use of graphics, images, and continuous media. This trend is especially apparent in software for synchronous distributed collaborative work, electronic meetings, virtual environments, telephony, interactive television, distributed multiperson games, etc. Software in these (and many other) domains manipulate sophisticated bitmap images, audio streams, video streams, and other data-intensive structures. A requirement that is common among these applications is that resources be available for a defined time period, i.e. there is some form of *deadline* and *service time* associated with each use of the resource. Further, the number of different deadline-sensitive resource requests varies dramatically with the way the user manipulates the computer; e.g, in a multimedia workstation environment, the resource usage pattern may vary with the activity of the user, often leading to phases of oversubscription for the resources and, as a result, missed deadlines. Because these environments allow the system to occasionally miss deadlines, such applications are often referred to as "soft real-time" applications.

Classic *hard real-time* systems support applications in which the work is defined in terms of repetitive phases, each phase having a prespecified service (or execution) time and a deadline [29]. In *periodic* hard real-time, the phases are of fixed length, and as soon as one period has completed, the next begins. Constructing periodic real-time software to operate properly can be very difficult, since the system must not admit work unless it can guarantee that the work (with a prespecified maximum service/execution time) will be performed after a specified release time and before a specified deadline. Even so, for certain well-defined problems, there are known procedures by which such a system can be built. Hard real-time becomes more daunting if the lengths of individual periods are allowed to vary, or if there may be an interval of "other" time between the end of one phase and the beginning of the next (so-called sporadic and aperiodic real-time) [29]. The problem of assuring correct real-time operation can quickly become intractable. A hard real-time system is correct and operational if it never misses a deadline -- informally we say it is deadline-dependent; a hard real-time system fails if a deadline is ever missed. The primary performance metric for hard real-time systems is trivial: The system either succeeds by meeting every deadline, or it fails.

Soft real-time applications are related to hard real-time in that the application requirements are also cast in terms of phases of work, execution time, and deadlines. However, they differ in that various aspects of the requirements can be "softened" according to application requirements. The softening may occur in that the execution is not strictly periodic; applications may sometimes use more service time than they specified; the system may fail to provide enough resources for the application to meet its deadline, etc. In contrast to hard real-time, soft real-time systems do not fail in the case of a missed requirement, they merely provide some form of degraded benefit. Clearly, any system that provides an environment in which there are deadlines, but where any aspect of hard real-time can be relaxed, supports some form of soft real-time computing. The general problem is that as soft real-time technology has developed, researchers have had a tendency to focus on certain aspects of soft real-time, while ignoring others. A solution for one aspect may not be a solution suitable for an application that needs a different perspective on soft real-time. Different kinds of application tasks require different kinds of requirement softening -- some want soft deadlines, some want soft periods or time between periods, some want soft service time, some want to vary the amount of work per period, etc. In particular, the metrics used to evaluate a strategy are dependent on the specific aspects that the system manipulates. Currently it is possible to compare the performance of different application loads on a given system, but without compatible metrics among systems it is difficult to compare the performance of two different systems.

Quality of service (QoS) *levels* have emerged as a discrete language with which applications can convey their resource needs to the resource manager. The soft real-time requirements are abstracted into a table relating the value of a computation to a level of resource allocation [30][28][27][1][2][22]. In essence, the application designer evaluates the soft real-time criteria that are important for that application, writes the application 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 can use the benefit-allocation information to allocate resources according to the amounts needed versus the global benefit that can be achieved across the applications. Since it is argued that QoS levels adequately reflect the spectrum of soft real-time characteristics, it is then possible to focus on the way applications use levels to reflect system performance.

We evaluate the behavior of the resource manager in terms of how it manages the QoS level profiles. From the application perspective, the resource allocation manager will provide resources corresponding to a given level (the level may be dynamic). The behavior of the system can be represented by metrics to represent the way the value change as each application executes. *Benefit* refers to the amount of value provided by the applications as it executes, and *instability* (we also informally refer to this as "stability") refers to the frequency and distance of benefit changes during the execution. Intuitively, benefit focuses on *maximizing* the level over time, and instability 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 instability. While these metrics were developed in the context of QoS levels, they apply equally well to any system that employs value functions to characterize soft real-time.

The benefit and instability metrics provide a concrete measure of soft real-time, and they also 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 a 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 that different level-choosing algorithms provide different kinds of performance. Intuitively, some algorithms provide higher benefit at the cost of instability, others provide higher stability at the cost of average benefit, and still others provide performance somewhere in between. 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. As a result of using the metrics, we discovered a single policy that had the best benefit and best instability for the policies we considered.

This paper formally defines the metrics and presents results showing the performance of different systems relative to these metrics with a variety of algorithms and application suites. Section 2 presents a discussion of related research. Section 3 summarizes soft real-time and QoS levels. Section 4 provides a detailed discussion of our soft real-time performance metrics. Section 5 presents quantitative performance results from running our system with a variety of algorithms and application suites. Finally, Section 6 presents conclusions we have drawn from this work.

2 Related Work

A *value function* plots the value of a result from a computation versus the time at which the result becomes available [13]. In hard real-time, there is no (or even negative) value if the result does not become available until after the deadline. In soft real-time, the value may degrade after the deadline, but the degradation may be represented by an arbitrary function. Burns generalized value functions so that they also addressed isochronous applications -- ones in which the result should not be produced too early nor too late [5]. Tokuda and Kitayama discretized value functions into tables, and coined the term QoS levels, in 1993 [30]. Subsequently, QoS levels have been refined and used by Rajkumar et al.[28], Nahrstedt [22], Abdelzaher et al. [1][2], Brandt, et al. [26][12][3] [4], and others. There have also been other researchers that have captured the notion of level-like computation, though not using QoS level profiles explicitly: Compton and Tennenhouse described a technique by which applications would coordinate their behavior to voluntarily shed resources for the general benefit of the system [7]. Liu, et al. introduced the idea of *imprecise computation* in which each application defines a required and an optional part; the required part performs work that must be done, and the required part increases the quality of the result [9]. Massalin and Pu developed *software feedback* to allow processes to dynamically its processing time according to the system load [19].

Value functions, and their generalization into QoS levels, have influenced some aspects of the way resource managers are designed to support soft real-time applications. Mercer, Savage, and Tokuda developed *processor reserves* as a mechanism for supporting soft real-time allocation of the CPU [20]. Processor capacity reserves were implemented in Real Time Mach (RT-Mach). Jones, et al, generalized the idea of reserves in the Rialto scheduler [14][15]. Other researchers have independently derived related mechanisms for supporting various aspects of soft real-time: Nieh and Lam built the SMART system to support continuous media application where CPU scheduling takes the *importance* of an application's request into account when it schedules the CPU [23][24]. In the MMOSS system, Fan considers application benefit by allowing each application to define a range of resources in which it can operate [8].

3 Soft Real-Time and QoS Levels

QoS level systems allow a soft real-time application to specify a set of algorithmic solutions to be executed based on the availability of resources. The algorithms are ordered according to the resources they require and (implicitly or explicitly) according to the quality of the output they provide. A common resource manager uses this information to allocate the available resources to the executing processes. Missed deadlines are used as an indication that one or more processes is using too much resources and consequently the allocations must be modified to fix the problem. QoS level systems can be used as a mechanism for managing the resources of application executing on top of soft real-time operating systems such as RT-Mach.

In our study we have created a middleware QoS level system called the *Dynamic QoS Manager* (DQM) to manage CPU allocation [3][4]. In this system, each application specifies a set of algorithmic modes in which it can execute, along with the computational requirements and the corresponding benefit of running in each mode. The DQM provides the mechanism for managing soft real-time by managing the level of each running application through a variety of QoS allocation algorithms. In our variant of QoS levels, each application is characterized by two numbers, the *maximum CPU usage* and the *maximum benefit*. The maximum CPU usage is the fraction of the CPU required to execute the application at its most intensive resource level (this could also be specified in a CPU-specific measure such as the cycle count). The maximum benefit of the application is a user-specified indication of the benefit provided by executing the application when running at its highest QoS level - analogous to application priority, importance, or utility in other systems. Each QoS level is characterized by three numbers, relative CPU usage, relative benefit, and period. Relative CPU usage specifies the fraction of the maximum CPU usage required by the execution level, relative benefit specifies the fraction of the maximum benefit provided by the QoS level, and period specifies the amount of time to allocate for each iteration of the algorithm. Relative values are specified because the maximum CPU usage number depends on the system on which the application is being executed and the maximum benefit will be user-specified, but the relationship between the levels is expected to be constant in most cases and specifiable at application development time. Implicitly we assume that while all of the algorithms correctly implement the desired application, the benefit of the result degrades with a decrease in execution level. An application can be executed at any of the levels, using corresponding resources with corresponding benefit.

The specific QoS levels and corresponding CPU usage and benefit are application dependent. The levels themselves are determined by the goals of the application developer. They incorporate algorithmic trade-offs that represent a range of output qualities. Traditionally, real-time and multimedia application developers make these trade-offs as a matter of routine, selecting the highest quality algorithm that fits within their known computational constraints. With QoS levels the developer can incorporate multiple algorithmic options and allow the system to dynamically determine which one is most appropriate at run-time. The CPU usage numbers are directly measured and represent the average CPU usage for each level of an application. The benefit numbers are determined by the application developer. The details of determining appropriate benefit numbers goes beyond the scope of this research. However, a very large body of research has been conducted in other domains to determine acceptable video frame rates, audio jitter and delay, etc. The results of this research apply directly to the problem of determining the relative benefits of different execution levels.

4 Performance Metrics

The goal of the soft real-time performance metrics is to provide a means to characterize and compare the performance of different applications and systems. Because of the varying characteristics of soft real-time, it is difficult to find concise metrics to represent the full spectrum of performance, then to use those metrics to compare the behavior of different application workloads on different soft real-time systems. Just as QoS levels have been shown to be a reasonable language by which an application can distill its soft real-time characteristics into a single profile, we now argue that they are an appropriate basis for new soft real-time performance metrics.[1] QoS level profiles define the importance of missed deadlines, execution time, and period, so the new metrics focus on the performance in terms of levels -- what benefit (based on level of execution) does an application execute in a given environment, and how does the level fluctuate over time?.

In order to measure the performance of QoS level systems, additional metrics need to be developed. The number and frequency of missed deadlines and the amount by which deadlines are missed do not adequately characterize the performance of these systems. In particular, QoS level systems attempt to eliminate missed deadlines by modifying the processing characteristics of the processes. This results in changes in both temporal and qualitative changes in the

1. We do not advocate that existing measures be abandoned, but rather, that these new measures be used in addition to the existing ones. It is still important to measure the success or failure of an application, the system utilization, throughput, etc.

algorithms being executed. The changes in the algorithms result in changes in the timing and quality of the processes' output.

The first metric we use is *benefit*, which characterizes the quality of the output of a process. Each process, *i*, is characterized by a user-specified benefit number B_i that indicates the importance of the process to the user relative to the other processes currently executing. Each QoS level, *j*, for an application is characterized by a relative benefit number B_{ij} that characterizes the percentage of the process' benefit provided by that level.

- At any time *t*, the instantaneous benefit of process *i* is $B_i(t) = B_i \times B_{ij}$, where *j* is the level at which process *i* is running at time t.
- During any time period $\Delta t = t_2 - t_1$, total benefit is defined as $B_i(\Delta t) = \int_{t_1}^{t_2} B_i(t)$. With QoS levels, the benefit curve from t_1 to t_2 is a step function with *m* distinct pieces p_i each lasting time d_i, and the integral is a discrete sum, $B_i(\Delta t) = \sum_{i=1}^{m} (B_i(t_1 + d_i - \varepsilon) \times d_i)$, where $\varepsilon < d_i$. We multiply by d_i to normalize for the different (and possible varying) application periods.
- During any time period $\Delta t = t_2 - t_1$, average benefit is simply $\left(\frac{1}{t_2 - t_1}\right) B_i(\Delta t)$.

The benefit for a suite of applications is defined similarly:

- The instantaneous benefit at time *t* of a suite of running applications is $B(t) = \sum_{i=1}^{n} B_i(t)$.
- During any time period $\Delta t = t_2 - t_1$, total benefit is for a suite of applications is

$$B(\Delta t) = \sum_{i=1}^{n} B_i(\Delta t) \quad .$$

- During any time period $\Delta t = t_2 - t_1$, average benefit for a suite of applications is $\left(\frac{1}{t_2 - t_1}\right) B(\Delta t)$

Benefit gives a good indication of the quality of the output of a process, but it is incomplete. In particular, moving from one level 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 $I_i(t) = \left|B_i'(t)\right|$. For applications with discrete levels, the instantaneous instability is infinite during the transition from one level to another and 0 at all other times.
- During any time period $\Delta t = t_2 - t_1$, instability is defined as $I_i(\Delta t) = \int_{t_1}^{t_2} I_i(t)$. As above, with QoS levels, the benefit curve from t_1 to t_2 is a step function with *m* distinct pieces p_i each lasting time d_i, and the integral is a discrete sum, $I_i(\Delta t) = \sum_{i=1}^{m} \left|(B_i(t_1 + d_i + \varepsilon) - B_i(t_1 + d_i - \varepsilon))\right|$, where $\varepsilon < \min(d_i, d_{i+1})$.
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 of application *i* from time t_1 to t_2 is $\left(\frac{1}{t_2 - t_1}\right) I_i(\Delta t)$.
- The instantaneous instability at time *t* of a suite of running algorithms is $I(t) = \sum_{i=1}^{n} I_i(t)$.
- The instability of a suite of running applications from time t_1 to t_2 is $I(\Delta t) = \sum_{i=1}^{n} I_i(\Delta t)$.
- The average instability of a suite of running applications from time t_1 to t_2 is $\left(\frac{1}{t_2 - t_1}\right) I(\Delta t)$

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 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.

5 Results

This section presents the results of running our system with various application suites. All of the experiments described in this paper were executed on a 200 Mhz Pentium Pro system running Linux 2.0.30. All applications and middleware were executed using the standard Linux scheduler. In order to generate data, two representative applications were used in various combinations. The applications are two different 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 above algorithms.

Figure 1 shows a representative set of benefit curves from executing the Proportional algorithm with the two applications described above. 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 DQM 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, $B(\Delta t)$ is 493.90 and $I(\Delta t)$ is 84.52. There were 6 missed deadlines out of 880 total application periods.

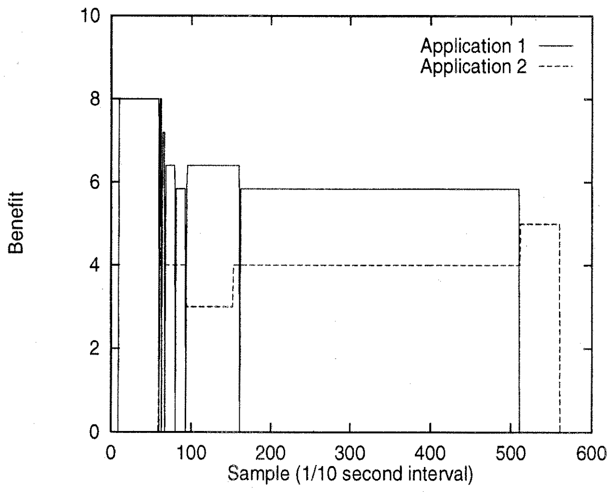

Figure 1: Benefit for Suite A with Proportional

In order to exercise the metrics we ran 3 different application scenarios 20 times with each of several 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%. Table 1 shows a summary of the average Benefit and Instability for the entire set of experiments. Each set of experiment showed roughly the same relative values and the averaged values serve to show the trends with respect to the metrics.

Table 1: Algorithm Comparison

Algorithm	Benefit	Instability
Calculated Benefit Optimal	588	3.27
Distributed	543	446
Proportional	552	207
Optimal Benefit	526	1809
Hybrid	541	942
Greedy (Benefit)	514	309
Greedy (Benefit Density)	514	324
Calculated Stability Optimal	560	0

The results from several different resource allocation algorithms are shown in the table:

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.

Distributed - a decentralized algorithm in which each application lowers its level whenever it has a certain number of deadline misses 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 [30]. This extremely simply algorithm did surprisingly well in terms of Benefit, with moderate instability relative to the other algorithms.

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 [23][24]. The main difference is that SMART is a missed-deadline based soft real-time system rather than a QoS level soft real-time system. This very simple distributed algorithm did the best with respect to both metrics.

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.

Hybrid - a centralized algorithm that begins with the optimal allocation, then employs a limited depth brute-force optimization to determine the resource allocation that maximizes overall benefit. In this variation of Optimal Benefit, each application is allowed to change by only 1 level per reallocation. 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.

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][2], albeit only in the context of the CPU resource. 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 even gets close to one.

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. It was hoped that this algorithm would provide better stability than Greedy Benefit while still maximizing benefit. The results show that this algorithm did no better than Greedy Benefit in terms of Benefit and did slightly worse in terms of Instability.

Calculated Stability Optimal - a simulated algorithm that calculates the highest obtainable benefit with 0 instability. This algorithm shows the best performance that can be expected assuming a priori knowledge of the applications and a requirement of 0 instability, analogous to hard real time performance. In the average, none of the algorithms performed as well as this a priori allocation. However, Proportional provided somewhat higher benefit in the 4 application experiment.

The results presented above are interesting for several reasons. Primarily, they show that they 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 obtainable benefit (Calculated Benefit Optimal). Within this small range, the simplest distributed 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.

With respect to soft real-time applications, the most important question is what kind of algorithm performance yield the best application performance. This question is also impossible to answer in general, although it is possible to specify the performance that is best for any particular application. Accordingly, we classify applications into 4 categories according to the kind of algorithm performance that results in good application performance: Hard Real-Time (HRT), Stability-Centric Soft Real-Time (S-SRT), Benefit-Centric Soft Real-Time (B-SRT), and best-effort or non-real-time (NRT). As we will discuss below, the application requirements all fall somewhere along the line between benefit performance and stability performance.

HRT applications are those applications that require complete stability, i.e., that have an instability of 0. These applications require that no application deadlines be missed and have no flexibility in the amount of work performed each application period. At least, the estimated amount of work per period is the worst-case estimate so that every frame is guaranteed to finish before its deadline. Benefit is simply a measure of the performance of the algorithm given the timing constraints. If additional benefit is required, the system is engineered so that the desired level of benefit can be stably provided. A television is a good example of a hard real-time system inasmuch as the hardware is required to keep up with the incoming stream of video frames and does the same amount of work per frame.

S-SRT applications are those whose performance is considered better if they are more stable, but benefit from taking advantage of additional resources as they become available. Such applications provide better performance given infrequent changes in the level of service, but can afford to do so occasionally. A good example of this is a desktop video stream such as is provided by an mpeg player. From the users perspective, it is better to run at stable rate of 20 frames/second than to jump back and forth every 5 seconds between 10 and 30 frames/second, even though the average benefit might be the same in both cases. Similarly, it would be better to show the video at 1/2 resolution than to jump back and forth from 1/4 to full resolution.

B-SRT applications are those whose performance is better given high benefit and can endure high but finite instability. Such applications still have deadlines, but whose performance is based more on average benefit than on stability. A visual tracking algorithm could be considered to be benefit-centric soft real-time inasmuch as infrequent but highly accurate tracking results are approximately as useful as frequent but less accurate tracking results.

NRT applications are those applications that care solely about benefit and for whom stability is not an issue at all. These applications are, in effect, best effort and are not necessarily real-time at all inasmuch as it doesn't really matter when the benefit is accumulated, just that it is accumulated at all.

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 different classes of applications 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.

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 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.

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.

6 Conclusion

The results of our limited 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, a lot more research with a wider range of application sets needs to be done before such a conclusion can be reached. In particular, it seems trivially obvious that a simple admission-based policy that assigns appropriate levels at admission time should easily be able to provide better performance with respect to Instability.

While our results deal specifically with our QoS level soft real-time system, we believe that the metrics will apply equally well to any soft real-time system. In particular, any such system that can provide an output benefit curve.

7 Bibliography

[1] T.Abdelzaher, E. Atkins, and K. Shin, "QoS Negotiation in Real-Time Systems and its Application to Automated Flight Control," *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium*, Jun. 1997.

[2] 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.

[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*, 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*, Dec. 1998, to appear

[5] A. Burns, "Scheduling Hard Real-Time Systems: A Review", *Software Engineering Journal*, May 1991.

[6] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole, "A Distributed Real-Time MPEG Video Audio Player", *Proceedings of the 5th International Workshop on Network and Operating System Support of Digital Audio and Video*, Apr. 1995.

[7] 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.

[8] 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.

[9] W. Feng and J. Liu, "Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines", *IEEE Transactions on Software Engineering*, Vol. 20, No. 2, Feb. 1997.

[10] H. Fujita, T. Nakajima and H. Tezuka, "A Processor Reservation System Supporting Dynamic QoS Control", *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, Oct. 1995.

[11] D. Hull, W. Feng, and J. Liu, "Operating System Support for Imprecise Computation", *Proceedings of the AAAI Fall Symposium on Flexible Computation*, Nov. 1996.

[12] 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.

[13] E. Jensen and 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.

[14] 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.

[15] 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.

[16] 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.

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

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

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

[20] 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.

[21] T. Nakajima and H. Tezuka, "A Continuous Media Application Supporting Dynamic QoS Control on Real-Time Mach", *Proceedings of the 2nd ACM International Conference on Multimedia*, pp. 289-297, 1994.

[22] K. Nahrstedt, H. Chu, S. Narayan, "QoS-aware Resource Management for Distributed Multimedia Applications", Journal on High-Speed Networking.

[23] 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.

[24] 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.

[25] G. Nutt, "Model-Based Virtual Environments for Collaboration", Technical Report CU-CD-799-95, Department of Computer Science, University of Colorado at Boulder, Dec. 1995.

[26] 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.

[27] 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.

[28] 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.

[29] J. Stankovic, M. Spuri, M. Di Natale, G. Buttazzo, “Implications of Classical Scheduling Results for Real-Time Systems”, *IEEE Computer* 28(6):16-25, 1995.

[30] 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.