Diverse Soft Real-Time Processing in an Integrated System

Caixue Lin, Tim Kaldewey, Anna Povzner, and Scott A. Brandt
Computer Science Department, University of California, Santa Cruz
{lcx,kalt,apovzner,scott}@cs.ucsc.edu

Abstract

The simple notion of soft real-time processing has fractured into a spectrum of diverse soft real-time types with a variety of different resource and time constraints. Schedulers have been developed for each of these types, but these are essentially point solutions in the space of soft real-time and no single scheduler has previously been offered that can simultaneously manage all types. More generally, no detailed unified definition of soft real-time has been provided that includes all types of soft real-time processing.

We present a complete real-time taxonomy covering the spectrum of processes from best-effort to hard real-time. The taxonomy divides processes into nine classes based on their resource and timeliness requirements and includes four soft real-time classes, each of which captures a group of soft real-time applications with similar characteristics. We exploit the different features of each of the soft real-time classes to integrate all of them into a single scheduler together with hard real-time and best-effort processes and present results showing their performance.

1 Introduction

Modern embedded, special-purpose and general-purpose computing systems are becoming increasingly complex and powerful. At the same time, the traditional notions of best-effort and real-time processing have fractured into a spectrum of processing classes with different time constraints including critical hard real-time applications, non-critical soft real-time applications and best-effort applications [5]. Within soft real-time, there is a large spectrum of applications with different types of softened time-constraints: meeting a minimum number of deadlines, requiring average resource utilization, adapting to available resources, *etc.*

Although it is recognized that a variety of different types of soft real-time exist with markedly different characteristics, no unified classification of all existing soft real-time applications has previously been presented. Most existing scheduling models [7, 3, 10, 12] therefore support only one or two classes of soft real-time processes. The result is that when faced with a new type of soft real-time constraint, a system must under- or over-constrain its processing, resulting in either reduced performance of the application, or reduced system utilization.

Our goal is a complete integrated real-time system supporting all types of real-time processing constraints. The Rate-Based Earliest Deadline (RBED) scheduler [5] previously integrated hard real-time, (one type of) soft real-time, and non-real-time processes. We now discuss the incorporation of the full range of soft real-time processes into RBED. We first present a complete real-time taxonomy which divides processes into nine different classes—including four soft real-time classes—based on their resource and timeliness requirements. The taxonomy fully characterizes the different processing and timeliness needs of the different types of real-time processes and allows us to support the different characteristics in a single system. Next, we discuss the integration of each of the soft real-time classes defined by the taxonomy into RBED and present experiments that validate the integration. The result is a single system and scheduler that natively support a complete range of real-time and non real-time processes, including adaptive and non-adaptive soft real-time, firm real-time, and rate-based processes in addition to hard real-time and best-effort.

2 Related work

Many different soft real-time processing models have been developed to support soft real-time scheduling in various environments [10, 2, 22, 12, 13, 15, 20, 9, 26, 3, 7]. In general, each of these models classifies all soft real-time applications into a single class by considering them having similar or same timeliness features. For example, (m,k)-firm real-time [10] and weakly-hard real-time [2] assume all applications have firm constraints which allow some number of deadline misses in every fixed-size window of job instances; Reservation-based models [22, 12, 13, 15] assume all applications require X units of processing over an interval of Y, thus a share X/Y of the CPU; Adaptive soft real-time models [20, 9, 26, 3] assume all applications have the ability to adapt their qualities to the available resources. Each of these soft real-time models captures one or at most

a few of the existing types of soft real-time applications, and none supports all the soft real-time classes. Our work differs in that we want to classify soft real-time applications into a minimum number of soft real-time classes, each capturing a group of soft real-time classes with similar features, and then support all of these classes.

Furthermore, some of these processing models only support the scheduling of their defined classes of soft real-time applications but not fully integrated scheduling of hard real-time or non-real-time (best-effort) applications. For instance, (m,k)-firm real-time [10] does not support hard real-time applications because it cannot guarantee all deadlines in overloaded conditions. HLS [28] is a hierarchical scheduler, which composes arbitrary hierarchies of existing schedulers in order to execute mixed class workloads. We want integrated scheduling of hard real-time and best-effort applications and all classes of soft real-time applications using a single scheduler (without the added complexity of understanding interactions of multiple schedulers).

3 Real-time taxonomy

Soft real-time tasks are conventionally defined as tasks with soft deadlines *i.e.*, deadlines that can be missed without compromising the integrity of the system [6]. While largely correct, such definitions fail to fully capture the diversity of the timeliness features found in various soft real-time systems and applications. For example, how many deadlines may be missed and by how much? If a deadline is missed, will the task continue to execute or abort? What if an application changes its processing so that it no longer misses deadlines, but provides lower quality output?

3.1 Soft real-time applications

There is a large variety of soft real-time applications. Some commonly used soft real-time applications and their timeliness features include:

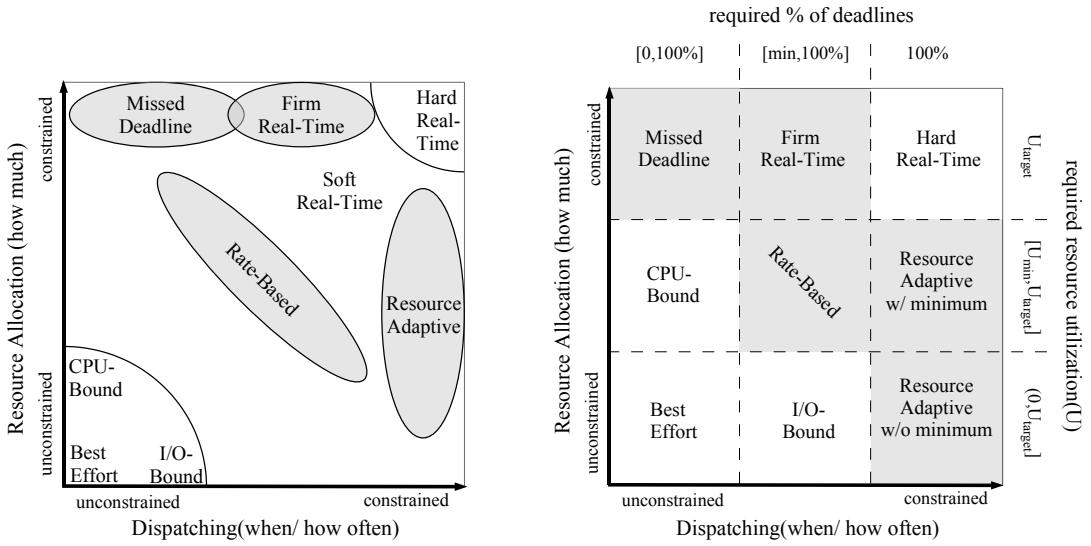
- Desktop and streaming audio [23]: No fixed deadlines, but require continuous processing at a fixed (average) rate.
- Desktop and streaming video (such as MPEG, RM, *etc.*): may adapt to the available resources and/or drop late frames [3].
- Virtual reality games [27] and interactive graphics: usually adapt to available resources and vary frame rate.
- Automatic control and monitoring systems [2]: fixed deadlines, but oversampling techniques allow occasional deadline misses.
- Adaptive control systems [21]: can adapt to available resources by executing with different sampling periods.
- Other examples: Simulations of physical systems/flight simulators, speech and image processing, soft modems, ...

Treating all of these different types of applications the same may fail to fully utilize the system resources or provide the best performance possible. Audio playback applications, for example, may be run as hard real-time processes, but doing so fails to take advantage of flexibility due to their ability to buffer decoded data during playback. Similarly, interactive games may drop frames, but better performance may be achievable by reducing color depth. Providing each application with exactly the resources and timeliness requirements that it needs will ultimately provide both better performance and greater flexiblity, but doing so requires a uniform model characterizing the varying requirements of these very different types of applications. Our real-time taxonomy does exactly this, unifying the different types of soft real-time applications into a single model based on their resource and timeliness needs.

3.2 The taxonomy

The Resource Allocation and Dispatching (RAD) model [5], conceptually depicted in Figure 1(a), separates the two aspects of resource management implicitly handled by all schedulers. It represents the diverse timing needs of various applications in terms of the degree of flexibility required with respect to resource allocation, or how much resources are required, and dispatching, or when the resources are required. Hard real-time (HRT) processes, for example, have extremely tight resource allocation and dispatching requirements: they must be guaranteed the resources required to execute for their worst-case execution time every period. Best-effort (BE) processes, by contrast, have very loose resource allocation and dispatching requirements, generally being able to run as slow and sporadically as necessary without being thought of as having failed. However, there is variation in terms of these requirements even within best-effort scheduling: non-interactive CPU-bound processes need greater amounts of CPU, but within very broad parameters they can use it in any size increments and at any time, while I/O-bound processes, especially interactive ones, use relatively little CPU but need to receive it quickly once they have unblocked in order to provide good interactive responsiveness.

Between hard real-time and best-effort lies the broad class of applications and systems referred to as soft real-time (SRT). This includes a variety of different systems with varying properties, all of which share the common property that resource allocation and/or dispatching requirements are looser relative to hard real-time. Figure 1(a) divides these into four broad sub-categories—Missed Deadline Soft Real-Time (MDSRT), Firm Real-Time (FRT), Resource Adaptive Soft Real-Time (RASRT) and Rate-Based



(a) RAD model and real-time classes (b) Real-time taxonomy

Figure 1. Real-time taxonomy

(RB)—depending upon which constraints are relaxed. MDSRT is real-time processing in which the time constraint is entirely softened such that some or even all deadlines may be missed by varying degrees [5, 16, 22, 23]. FRT is real-time processing in which the time constraint is relatively softened (compared to hard real-time processing) such that a specified number of deadlines may be missed in a given window of job instances [10, 2]; jobs whose deadline has been missed are considered invalid and are dropped. By contrast, RASRT is real-time processing in which the resource allocation constraint is softened by adapting application processing requirements to the available resources while attempting to minimize the number and amount by which deadlines are missed [3, 4, 16, 29]. Finally, in Rate-Based processing both resource allocation and dispatching can vary, but not completely independently: if more resources are provided a longer time may elapse before resources are once again allocated, and vice versa [25, 12].

The conceptual diagram of Figure 1(a) is formalized in Figure 1(b). The X-axis defines three degrees of dispatching constraints (from softest to hardest): 1) a task may be given the resources it requires at any time *i.e.*, all jobs may miss their deadlines, 2) a minimum percentage of the jobs of a task must meet their deadlines[1]; and 3) all jobs of a task must meet their deadlines. The Y-axis similarly defines three degrees of resource allocation constraints (from softest to hardest): 1) a task may receive any resource rate up to and including its target resource rate *i.e.*, it has no minimum resource rate, *i.e.*, $U_s \in (0, U_t]$; 2) a task may receive any resource rate between its minimum U_{min} (U_m for short) and its target resource rate U_t, *i.e.*, $U_s \in [U_m, U_t]$; and 3) a task must receive a resource rate U_s equal to its target (worst-case) resource rate U_{target} (U_t for short), *i.e.*, $U_s = U_t$.

This yields nine different classes of resource and timeliness requirements: 1) True best-effort with no processing or timeliness requirements; 2) CPU-bound best-effort, with some processing requirements; 3) I/O-bound best effort, with some timeliness requirements; 4) Missed-Deadline Soft Real-Time, with fixed soft deadlines; 5) Firm Real-Time, with some combination of soft and hard deadlines; 6) Resource-Adaptive Soft Real-Time, with the ability to adapt to any degree of resource availability; 7) Resource Adaptive Soft Real-Time, with a minimum amount of resources required; 8) Rate-based, with a required average resource rate, but flexibility both in the timing and amount of resources required at any given time[2]; and 9) Hard Real-Time.

The previous version of RBED supported BE, one type of SRT, and HRT. The rest of this section discusses how to exploit the timeliness features and properties of the various types of soft real-time tasks defined by the taxonomy.

3.3 Missed-deadline soft real-time

Missed-deadline soft real-time is the basic non-adaptive soft real-time used by many of the scheduling frameworks in the literature [16, 22, 23, 5]. Missed-deadline soft real-time tasks can miss some or all of their deadlines during overload. Missed-deadline jobs may complete late or be dropped, depending upon the application. A missed deadline soft real-time task has a target resource rate (U_t), less than or equal to its worst case resource rate. Since a missed deadline soft real-time task may miss all of its deadlines, it

[1] We do not distinguish the case where all deadlines may be missed by a fixed maximum amount, as this is equivalent to meeting 100% of appropriately determined pseudo-deadlines.

[2] The center of the diagram also include Resource Adaptive Soft Real-Time with soft deadlines, which combines the characteristics of both RASRT and MDSRT and has softer requirements than rate-based processing.

has a zero minimum resource rate ($U_m = 0$). Missed deadline soft real-time tasks include all soft real-time applications and systems that do not have the ability to adapt their qualities to the available resources.

3.4 Firm real-time

Firm real-time tasks can miss some, but not all, deadlines ($min > 0$) when resources are limited. In firm real-time processing, jobs that are going to miss or have missed their deadline are usually dropped because late results have little or no value. For example, it is often better to skip a frame in video playback than to display it late. Firm real time tasks are considered hard real-time for the purposes of admission control. Examples of firm-real time applications include video applications and computer-driven automatic control and monitoring systems using oversampling techniques [2].

The literature distinguishes two types of firm real-time processing: statistical firm real-time and pattern-based firm real-time. A statistical firm real-time task allows a certain percentage of its jobs to miss their deadlines, but limits the number of consecutive deadline misses. A (mr, mn)-firm task allows mr percentage of jobs to miss their deadlines as long as the number of consecutive deadline misses does not exceed mn. Pattern-based firm real-time tasks allow to drop jobs in user or system defined patterns, such as (m,k)-firm [10] also referred to as weakly-hard real-time [2]. A (m,k)-firm task requires at least m out of every k of its jobs to meet their deadlines. A (m,k)-firm constraint is usually stricter than a (mr, mn)-firm constraint for small k, e.g. $k <= 100$. Based on this assumption, (mr, mn)-firm tasks can be converted to (m,k)-firm with $k = \lceil \frac{100 \cdot mn}{mr} \rceil$ and $m = k - mn$ without loosening constraints. Therefore it is sufficient to describe firm real time processing as (m,k)-firm in an integrated system. Firm real-time applications often have additional constraints for their tasks, for example some jobs may be more important than others and should not be dropped (*e.g.*, MPEG I frames).

3.5 Resource adaptive soft real-time

Resource adaptive soft real-time applications adapt their resource usage and therefore their Quality of Service (QoS) based on the available resources. For example, a video stream server may gracefully reduce the video quality in situations of overload by adjusting the pixel density or frame rate of the served video stream. In general, applications may change their sampling interval, frame rate, bit rate, display size, compression algorithm, or any other algorithmic parameter affecting resource usage. Resource adaptive soft real-time tasks can be divided into sub-categories depending upon whether they adapt continuously or via discrete QoS levels and whether or not they have a minimum resource rate. An example of resource adaptive soft real-time with discrete QoS levels is adaptive control systems [21], where a control task is allowed to use different sampling periods. An example of resource adaptive soft real-time with continuous QoS levels is a chess program designed to play in a tournament with a clock: more time will result in a better move, but the program will always output a move in the available time.

Resource adaptive soft real-time tasks have a target resource rate U_t, which limits the maximum resource rate tasks can receive. Whether continuous or discrete, the different algorithms corresponding to the different resource allocations are implicitly or explicitly associated with *benefit* values [6, 3]; higher values are associated with larger resource allocations. For example, for adaptive control systems, the benefit may be an *instantaneous cost function* [21] or a *finite-horizon cost function* [11] of the control tasks.

We assume that each resource adaptive soft real-time task has a worst-case resource requirement—which may change as resource allocations change—so that the task is always guaranteed to meet all of its deadlines if it adapts to the available resources. The difference between resource adaptive and missed deadline soft real-time tasks is that missed deadline soft real-time tasks miss deadlines when resources are reduced while resource adaptive soft real-time tasks voluntarily adapt their performance to the available resources. Hybrids are also possible, where some deadline misses are allowed but adaptation takes place if too many occur in a given time window.

3.6 Rate-based

Rate-based tasks have constraints in the form of continuous processing requirements (U_m) [25]. A rate-based task usually has a buffer to hold the produced data temporarily. This allows rate-based tasks to be very flexible about how much and when resources are needed. Frequent, small allocations of resources may be used, as may infrequent, large allocations, or any combination of the two, as long as the buffer never underflows or overflows. If more space is filled in the buffer a longer time may elapse before the buffer is replenished, and if less space is filled, less time may elapse. Larger buffers (*e.g.*, the amount of RAM in an audio card) provide more flexibility. A rate-based task with no buffer capacity degenerates to a hard real-time task.

Any process that uses a buffer or a queue to communicate with another process or a device may be considered a rate-based task. In an audio player, a rate-based task is a process that reads audio data frames, decodes and writes them to a fixed-size memory buffer, and they are consumed from the buffer by the sound card driver at a constant rate. Other examples include real-time video recording, such as burning a VCD/DVD, where a fixed buffer is used to control the burning process.

The flexibility of rate-based tasks allows them to take ad-

vantage of idle time in the system and, in effect, buffer execution time for other tasks in the system. When the system is lightly loaded, a rate-based task can temporarily run at a faster rate and keep the system busy, as long as the buffer does not overrun. When the system is heavily loaded, a rate-based task can temporarily run at a slower rate as long as the buffer does not underrun. In section 4, we discuss different ways to effectively exploit the flexibility of rate-based tasks to improve the overall performance of our integrated system.

4 Integrating diverse soft real-time scheduling into RBED

RBED is an integrated scheduler supporting hard real-time, soft real-time and best-effort processes. It allocates resources to processes as a percentage of CPU such that the total allocated rate is less than or equal to 100%, and then schedules all processes with EDF, using timers to enforce resource allocations. RBED dynamically changes allocated resources and application periods without violating EDF constraints, guaranteeing that tasks never miss their assigned deadlines. Previous implementations of RBED treated all soft real-time applications as missed-deadline soft real-time. Below we describe the integration of missed-deadline soft real-time and the remaining three soft real-time classes into RBED.

We implemented each soft real-time class in the RBED scheduler [5] in the Linux 2.6 kernel. For our experiments we used a 1 GHz Intel Pentium III machine. All real-time workloads used in the experiments were generated by a tool we developed for this purpose. As input it takes a period or minimum inter-arrival time, a worst-case or average-case execution time, and the desired process type. Based on these parameters it generates periodic hard real-time or soft real-time tasks with variable execution times in either a normal (NW) or a left half-normal (NA) distribution [19]. Since we focus on the performance of real-time applications in a mixed environment, we arbitrarily reserve a minimum of 2% of the CPU for best-effort tasks, enough to provide a functional interactive system for running command shells during the experiments.

4.1 Missed-deadline soft real-time

We use weighted proportional share to allocate resources to missed deadline soft real-time tasks in our system. Assuming the total rate of all hard real-time tasks is U_s^{HRT}, and the reserved minimum resource rate for all best-effort tasks is β, the maximum available resource rate for all missed-deadline soft real-time tasks is $U_{SRT} = 1 - \beta - U_s^{HRT}$. When the system is underloaded—the total target rate is less than the total available rate: $\sum_{j=1}^{N} U_{t,j} <= U_{SRT}$, where N is the number of missed-deadline soft real-time tasks—a missed-deadline soft real-time task T_i is allocated a resource rate U_s equal to its target rate: $U_s = U_t$. When the system is overloaded—$\sum_{j=1}^{N} U_{t,j} > U^{SRT}$—a proportional fair share resource allocation policy is used to share resources among missed-deadline soft real-time tasks. That is, T_i will be allocated a rate proportional to its target rate: $U_s = \frac{U_{t,i}}{\sum_{j=1}^{N} U_{t,j}} \times U_{SRT}$. Each missed-deadline soft real-time task may have a weight (W) to denote its right to get resource share relative to other tasks. In this case, the proportional resource allocation for a missed-deadline soft real-time task T_i with weight W_i should be adjusted as $U_s = \frac{W_i \times U_{t,i}}{\sum_{j=1}^{N} (W_i \times U_{t,j})} \times U_{SRT}$. This resource allocation mechanism is similar to the traditional proportional fair share algorithms, *e.g.* WFQ [8] and lottery scheduling [30].

Once resource allocation is done, each missed-deadline soft real-time task is guaranteed to receive its allocated resources, although its budget and period (B_s, P_s) may differ from its target ones (e, p). Currently we adjust the received budget and period as $(B_s = e, P_s = \frac{B_s}{U_s})$ by extending the period while keeping the budget. With this adjustment, a missed-deadline soft real-time task may frequently miss its deadlines in overload situations. Furthermore, some missed-deadline soft real-time tasks may even miss deadlines in underloaded conditions because their target resource rates may not reflect their actual resource rates. Since a missed-deadline soft real-time task does not have the functionality to lower its requirement when missing the current deadline, its current job continues to run until completion, at which the next job will be released. As a result, it appears to run "slower" than expected in overload situations.

4.1.1 Evaluation

We implemented missed-deadline soft real-time processing in RBED using proportional fair sharing resource allocation. The default weight of each missed-deadline soft real-time task is set to 1. A system call is implemented to allow dynamic changes of task weights.

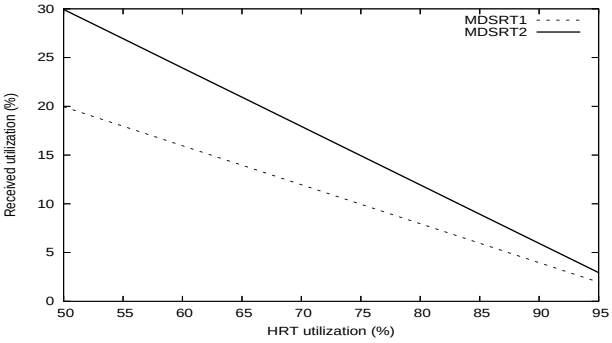


Figure 2. Proportional share among MDSRT tasks

The first experiment shows the proportional resource share among multiple missed deadline soft real-time tasks

with their weights equal to 1. The workload consists of one HRT task and two missed-deadline soft real-time tasks (MDSRT1 and MDSRT2). MDSRT1 and MDSRT2 request 30% and 20% of CPU respectively and the HRT task has varying resource utilization ranging from 50% to 95%. Figure 2 shows the received utilization of the two missed-deadline soft real-time tasks as the utilization of the HRT task increases. The utilization of the two missed-deadline soft real-time tasks decreases exactly proportional to their requested resource utilization.

The second experiment shows the weight impact on the relative performance of missed-deadline soft real-time tasks using proportional fair share. The workload consists of two missed-deadline soft real-time tasks: MDSRT1 (75%, e=75ms, p=100ms) and MDSRT2 (75%, e=150ms, p=200ms). Figure 3 shows the deadline miss ratio of MDSRT1 and MDSRT2 as their relative weight ($\frac{W_2}{W_1}$) changes. MDSRT2 achieves better performance—in terms of smaller deadline miss ratio—by trading off MDSRT1 performance as the relative weight increases.

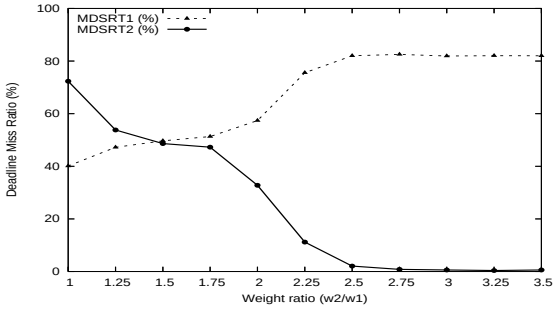


Figure 3. Weight impact on performance

4.2 Firm real-time

Firm real-time tasks are treated as hard real-time for the purpose of admission control. That is, a firm real-time task T with target (worst-case) resource rate U_t will be allocated a rate $U_s = U_t$. Therefore, a firm real-time task will not miss any deadlines if no job is dropped. However, in order to benefit other soft real-time tasks, actual resource usage of a firm real-time task may be less than allocated due to job dropout at runtime.

Drop mode defines the dropout pattern for a firm real-time task. A drop mode may be static or dynamic. A static drop mode picks the jobs to drop in a pre-defined way, such as early dropout (deeply red [18]) and evenly (uniform dropout [24]), *etc.* Early dropout always drops the first $m-k$ jobs and execute the remaining m jobs of a (m,k)-firm task in each static window of k jobs. For example, dropping the 1st, 2nd, 7th, 8th, ..., (6n+1)th, (6n+2)th, ... job of a (4,6)-firm task will not violate its constraint. Even dropout always drops jobs evenly as long as execution sequence does not violate (m,k)-firm constraint. For example, dropping the 1st, 4th, ..., (3n+1)th, ... job of a (4,6)-firm task will not violate its constraint.

A static drop mode does not prevent a firm real-time task from dropping jobs, even if no other tasks require additional resources. For example, a firm real-time task will continue to drop its jobs in underloaded states, hurting its own performance without benefitting any other task.

Our dynamic drop mode drops jobs on demand—when other tasks request additional resources—as long as the (m,k)-firm constraint is not violated. Dynamic drop mode captures the load status (under- or over-loaded) and the highly variable resource requirements of soft real-time (and best-effort) tasks in the system. By doing so, it ensures that firm real-time tasks achieve maximum performance when no other tasks require additional resources. In order to be able to drop firm real-time jobs dynamically without violating the (m,k)-firm constraints, a sliding window mechanism [10] needs to be applied. A sliding window mechanism tracking the last $k-1$ executions allows the system to determine whether the current job of a firm real-time task can be dropped without violating the given constraints.

4.2.1 Evaluation

We implemented (m,k)-firm real-time processing [17] in the RBED scheduler. The experiments show the influence of static and dynamic drop modes on performance of other soft real time tasks in a loaded system. Starting from a static early drop pattern, always dropping the maximum allowed number of consecutive jobs, we compare these results with evenly distributed and drop on demand patterns.

The workload detailed in Table 1 is used to evaluate the static early drop pattern. It consists of two hard real-time processes together accounting for 20% of CPU utilization, a firm real-time process with $(m,5)$ constraints and 28% of CPU utilization, and one missed-deadline soft real-time process with 50% of CPU utilization. We measure the influence of the firm real-time parameter m on the performance of the missed-deadline soft real-time processes in terms of deadline misses. Using the same firm real-time parameters, we investigate how the period length of the missed-deadline soft real-time task influences its deadline misses, while keeping its utilization constant.

Figure 4 shows the deadline miss ratio of the missed-deadline soft real-time task as a function of its period. A set of four curves (FRT-m) shows the missed-deadline soft real-time performance for the case that $m = 5,4,3,2$ consecutive jobs of the firm real-time task are required to meet their deadline while the remaining $k-m = 5-m$ ones are dropped. We consider the first curve (HRT = FRT-5) as a baseline measure, scheduling our firm real time task as hard real time without any dropped jobs. This experiment shows that static firm real-time drop patterns always result in equal or lower deadline miss ratio for other soft real-time tasks, compared to firm real-time tasks scheduled as hard real time. The results also indicate a quantitative relation-

ship between firm real-time and other soft real-time tasks: missed-deadline soft real-time performance improves as the number of dropped jobs increases from 0 to 3.

Table 1. Workload 1 (unit in *ms*)

Task	Task Parameters		Server Parameters			Adjustment	
	$e = f(\bar{e})$	p	$B = \bar{e}$	$P = p$	$U = \frac{B}{P}$	$\Delta(\bar{e})$	$\Delta(p)$
HRT	NW(20)	200	20	200	10%	0	0
HRT	NW(30)	300	30	300	10%	0	0
FRT (m,5)	NW(28)	100	28	100	28%	0	0
MDSRT	NA(25)	50	25	50	50%	+20	+40

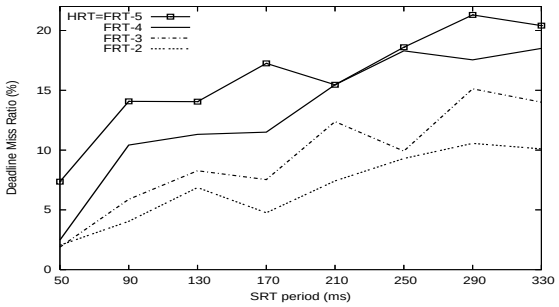


Figure 4. Static dropout impact on MDSRT performance (MDSRT: $u = 50\%$)

Table 2. Workload 2 (unit in *ms*)

Task	Task Parameters		Server Parameters			Adjustment	
	$e = f(\bar{e})$	p	$B = \bar{e}$	$P = p$	$U = \frac{B}{P}$	$\Delta(\bar{e})$	$\Delta(U)$
MDSRT	NA(10)	100	10	100	8%	+2	+2%
FRT(7,10)	NW(45)	100	45	100	45%	-1	−1%
FRT(7,10)	NW(45)	100	45	100	45%	-1	−1%

We further investigate the improvements that can be achieved by dropping firm real-time jobs evenly distributed and dynamically. Our dynamic dropout mode drops jobs if there is a demand for resources by other tasks, otherwise it drops jobs evenly. We compare the performance of all three dropout modes: early, evenly and on-demand. The sample workload, detailed in Table 2, consists of two firm real-time tasks, both having $(m,k) = (7,10)$ firm constraints and starting at 45% CPU utilization and a missed-deadline soft real-time task starting at 8% CPU utilization. To investigate the performance differences between the three firm real-time dropout modes across varying workloads, we increase the missed-deadline soft real-time task load stepwise by 2% and reduce the load of each firm real-time task by 1% correspondingly.

Figure 5 shows the deadline miss ratio of the missed-deadline soft real-time task under the three different drop modes as its own load increases. Compared to early drop mode, evenly drop always performs better or equal, while the dynamic drop mode achieves the best performance.

4.3 Resource adaptive soft real-time

As detailed in section 2, there are many resource adaptive soft real-time task models, including imprecise computation [20], QRAM [26], DQM [3], and [9] , all of which try

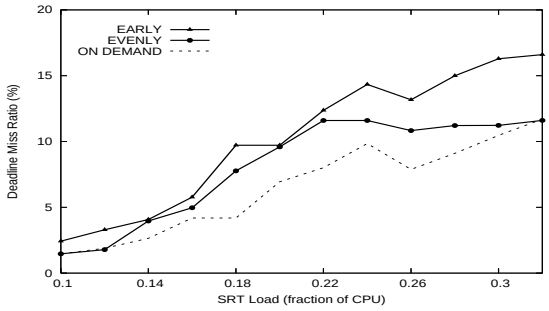


Figure 5. FRT dropout impact on performance

to maximize the global benefit or value that the system can achieve. It is NP-hard in general to optimize the overall benefit when resource adaptive applications with discrete QoS levels coexist in the system [20, 3], so heuristics are used to achieve high overall utility by dynamically adjusting the QoS levels of each soft real-time task given the available resources in the system. Our heuristic algorithm iteratively increases the level of a task which will provide the greatest increase in benefit density [14, 3], *i.e.*, the one with the greatest $\frac{\Delta benefit}{\Delta rate}$, until no more increases are possible within the available resources. Similarly, when lowering resources it always chooses the level whose removal decreases overall benefit density the least. Although the heuristic algorithm frequently finds the resource allocation that provides the highest possible benefit, this is not always guaranteed, nor is it always possible.

4.3.1 Evaluation

We implemented the resource adaptive soft real-time processing using heuristic resource allocation in RBED. A system call is implemented to allow the application to adjust its benefit (similar to the weight adjustment in missed deadline soft real-time processing) and query any resource adjustment done by the scheduler in the kernel. We expect a better resource adjustment communication mechanism by using signal notification in the future. The resource adaptive processing model and its implementation in RBED were validated with a real case study applied to adaptive control systems [21]. Here we present some basic experiments that demonstrate how the heuristic algorithm works.

The experiment shows the resource adaptation results as the offered workload changes. The workload consists of one HRT task with CPU utilization of 60%, and three resource adaptive soft real-time tasks (RASRT-1, RASRT-2, RASRT-3) with discrete QoS levels shown in Table 3. Figure 6 shows the changes of the QoS levels of RASRT-1. RASRT-2 and RASRT-3 as the HRT task enters and leaves the system. Initially, the HRT task uses 60% of the resource, and executes for 46.2 seconds. Therefore the maximum available resource rate for the resource adaptive soft real-time

Table 3. Benefit Tables (RASRT-1, RASRT-2 and RASRT-3 in Figure 6)

Number of QoS Levels: 4			
Level	Benefit	Rate	Period
1	**1.0**	**0.35**	**100 ms**
2	0.7	0.30	100 ms
3	0.5	0.20	100 ms
4	0.3	0.10	100 ms

(a) RASRT-1

Number of QoS Levels: 4			
Level	Benefit	Rate	Period
1	**1.0**	**0.45**	**100 ms**
2	0.8	0.40	100 ms
3	0.6	0.30	100 ms
4	0.4	0.10	100 ms

(b) RASRT-2

Number of QoS Levels: 4			
Level	Benefit	Rate	Period
1	1.0	0.60	100 ms
2	0.9	0.50	100 ms
3	0.7	0.40	100 ms
4	**0.5**	**0.10**	**100 ms**

(c) RASRT-3

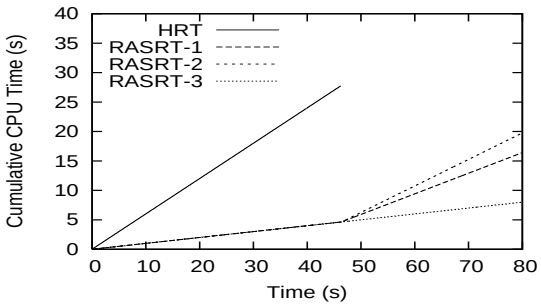


Figure 6. RASRT behavior in RBED

tasks is $100\% - 60\% - 2\% = 38\%$ (Note that we reserve 2% for all best-effort tasks in the system). As a result, the resource adaptive soft real-time tasks each receives 10%, their lowest QoS levels. When the HRT task leaves, the levels of RASRT-1, RASRT-2 and RASRT-3 are adjusted to provide a higher level of benefit within the available resources. The result is that RASRT-1 increases to level 1, RASRT-2 increases to level 1, and RASRT-3 remains at level 4, shown as the bold rows in Table 3.

4.4 Rate-based

Our integrated system currently treats rate-based tasks as hard real-time tasks when allocating resources and assigns $U_s = U_t$. There are three approaches to scheduling rate-based tasks: as periodic hard real-time, as a continuously releasable hard real-time, and as a continuously releasable hard real-time with blocking.

4.4.1 Rate-based as periodic hard real-time

In the first approach, rate-based tasks are scheduled as periodic hard real-time tasks. For both producer and consumer processes, we choose a block size of n bytes which is no greater than half of the buffer and no smaller than the minimum amount of data that can be requested by the consumer. For a producer process, the budget B_s is set to the worst-case time to produce n bytes and the period P_s is set to the time to consume n bytes. When each period ends, the previously produced n bytes are consumed, but the scheduler guarantees that next n bytes are produced by this time. The situation is reversed for consumer processes (those on the other side of the buffer). Thus, the algorithm guarantees there will be no buffer underrun provided that n exceeds the lowest required block size. To prevent buffer overrun, the maximum value of n is the half size of the buffer, because we need $2 * n$ bytes of unused space in the buffer to pre-fill it with n bytes and then produce another n bytes in the first period. The algorithm allows any choice of budget $B_s \in [e_{min}, e_{max}]$ and thus the corresponding period $P_s \in [\frac{e_{min}}{U_s}, \frac{e_{max}}{U_s}]$, where e_{min} is the worst-case execution time (WCET) needed by the task to produce the minimum amount of data requested by its consumer, and e_{max} is the WCET needed by the task to produce the amount of data that exactly fills half of the buffer.

The performance of rate-based processes is guaranteed by worst-case resource reservations, but overconstrains the processing, fails to take advantage of the flexibility inherent in rate-based processes, thereby limiting its ability to both produce and consume slack, and ultimately limits the overall performance of the system, as will be discussed below.

4.4.2 Rate-based as continuously releasable hard real-time

In this approach, a rate-based task T immediately releases its next job (if there is one) once it completes the current job, and the time needed to empty the buffer is assigned as the new deadline. If the buffer is full, the task will be blocked until some of the data in the buffer is consumed. In particular, the budget and period of T are assigned the minimum values: $(B_s, P_s) = (e_{min}, \frac{B_s}{U_s})$. Once T completes a current job, its new job is released immediately with the deadline assigned its previous deadline plus the period: $d_{s,j} = d_{s,j-1} + P_s$.

Similar deadline extension mechanisms are used in other systems [1, 19] to allow the current job to borrow from the budget reserved for the task's next job. Since the borrowed resources are executed with the deadline of the following job, the task's utilization remains unchanged and therefore the correctness of the scheduling is preserved.

With the deadline extension mechanism, a rate-based task automatically takes advantage of any available slack (allocated but unused resources) in the system to complete as many jobs as possible, and idles when the buffer is full, producing slack for other processes. Therefore, we expect this approach to be more efficient than the first one. However, this approach still fails to achieve the full potential of rate-based processes.

4.4.3 Rate-based as continuously releasable hard real-time with blocking

This approach extends the previous approach by also considering demands of other tasks in the system. If there are other tasks that need additional resources, a rate-based task T will block immediately until some minimum amount of data in the buffer is consumed or a minimum level of the buffer is reached. This sheds the total load and gives other tasks a better opportunity to use idle time in the system. Since this approach takes the dynamic resource usage into consideration and sheds load by blocking rate-based tasks if necessary, we expect better responsiveness of aperiodic soft real-time or interactive I/O bound best-effort processes.

4.4.4 Evaluation

We have implemented the three approaches for scheduling rate-based tasks in RBED and present experimental results of their relative performance. The workload, detailed in Table 4, consists of one HRT task with CPU utilization of 20%, one rate-based task (RB) with CPU utilization of 50% and the buffer size of $8 * n$, where n is the amount of data generated by each completed job, and one missed-deadline soft real-time task (MDSRT) with CPU utilization of 30%. We compare the performance of the missed deadline soft real-time task in the presence of the rate-based task scheduled respectively as a hard real-time task with period of 50ms, a continuously releasable hard real-time task (RB w/o block), and a continuously releasable hard real-time task with blocking (RB with block). We also varied the period of the missed deadline soft real-time task by changing its execution time while keeping its demanding utilization constant.

Table 4. Workload 3 (unit in *ms*)

Task	Task Parameters		Server Parameters			Adjustment	
	$e = f(\bar{e})$	p	$B = \bar{e}$	$P = p$	$U = \frac{B}{P}$	$\Delta(\bar{e})$	$\Delta(p)$
MDSRT	NA(60)	200	60	200	30%	+3	+10
RB	NW(25)	50	25	50	50%	0	0
HRT	NW(160)	800	160	800	20%	0	0

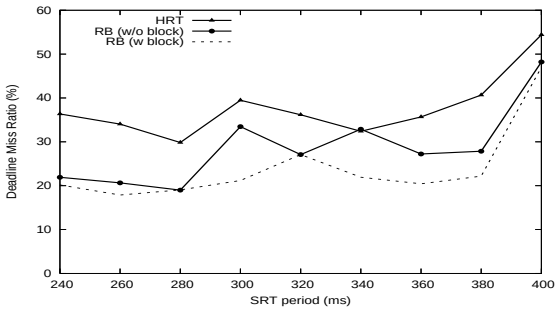


Figure 7. Rate-based scheduling behavior impact on performance

Figure 7 shows the deadline miss ratio of the missed deadline soft real-time task as its period increases under the three different approaches. The results show that the approach of scheduling rate-based tasks as continuously releasable hard real-time tasks with blocking outperforms the approach of scheduling rate-based tasks as a continuous releasable hard real-time tasks without blocking; Both outperform the approach of scheduling rate-based tasks as periodic hard real-ime tasks in all scenarios.

5 Conclusions and Future Work

The spectrum of real-time applications with different time and resource constraints requires a classification to optimally support each of them. In order to optimize system performance in an integrated real-time system, it is necessary to manage all classes of real-time applications together with best-effort applications, uniformly using a single scheduler. Based on the RAD model, we have developed a real-time taxonomy which captures the timeliness features of existing real-time applications. This is the first complete taxonomy that fully captures the range of real-time processing requirements from best-effort to hard real-time. Since best effort and hard real time processing have been extensively investigated in the past, we focused on the wide spectrum of soft real time classes, namely missed-deadline soft real-time, resource adaptive soft real-time, firm real-time and rate-based. Exploiting the timeliness features of each of these classes allowed us to describe the complete space of real-time processing and to integrate them into our RBED scheduler.

The promising performance results suggest an obvious next step, which is to develop a uniform resource allocation model to uniformly manage all classes of processes. Looking at the individual ways each soft real-time class is currently handled, it becomes obvious that not all processing mechanisms are fully compatible in a uniform system. For example, missed-deadline soft real-time tasks seek fairness through proportional share resource allocation, while resource-adaptive soft real-time tasks use a heuristic resource adaptation algorithm to achieve the highest possible global benefit. Assigning a benefit value to missed-deadline soft real-time tasks or giving up the benefit value for resource-adaptive soft real-time tasks makes satisfying their individual goals impossible. On the other hand, firm real-time and rate based tasks are both handled like hard real-time with the exception of allowing for certain missed deadlines or trade offs between resource allocation and dispatching constraints respectively.

Unifying these different processing mechanisms can be described in two ways. First, reduce the number of classes by either relaxing the requirements of more constrained classes, e.g. firm real-time such that it becomes missed-deadline soft real-time, or by increasing the requirements of less constrained classes, e.g. missed-deadline soft real-time

such that it becomes firm real-time. This approach, used by most existing schedulers, may result in reducing the number of classes into only one single class, which is not a flexible solution as explained in section 1. Second, merge the processing mechanisms of several different classes by simplifying their existing timeliness features or introducing new timeliness features. This approach will maintain the diversity of classes but unify the processing mechanisms for as many classes as possible. We favor the second approach and are currently investigating its implementation in the RBED scheduler.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, pages 4–13, Dec. 1998.

[2] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, Apr. 2001.

[3] S. Brandt and G. Nutt. Flexible soft real-time processing in middleware. *Real-Time Systems*, 22:77–118, 2002.

[4] S. Brandt, G. Nutt, T. Berk, and J. Mankovichr. A dynamic quality of service middleware agent for mediating application resource usage. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, pages 307–317, Dec. 1998.

[5] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, pages 396–407, Dec. 2003.

[6] A. Burns. Scheduling hard real-time systems: A review. *Software Engineering Journal*, 6:116–128, May 1991.

[7] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, Mar. 2002.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of the ACM SIGCOMM Symposium*, pages 1–12, Sept. 1989.

[9] C. Hamann, J. Löser, L. Reuther, S. Schönberg, J. Wolter, and H. Härtig. Quality assuring scheduling-deploying stochastic behavior to improve resource utilization. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, Dec. 2001.

[10] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, Apr. 1995.

[11] D. Henriksson and A. Cervin. Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In *In Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2005)*, Dec. 2005.

[12] K. Jeffay and D. Bennett. A rate-based execution abstraction for multimedia computing. In *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, Apr. 1995.

[13] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS 1999)*, pages 304–314, Dec. 1999.

[14] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the 6th IEEE Real-Time Systems Symposium (RTSS 1985)*, Dec. 1985.

[15] M. B. Jones, J. S. B. III, A. Forin, P. J. Leach, D. Roşu, and M.-C. Roşu. An overview of the Rialto real-time architecture. In *Proceedings of the 7th ACM SIGOPS European Workshop*, pages 249–256, Sept. 1996.

[16] M. B. Jones, D. Roşu, and M.-C. Roşu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, pages 198–211, Oct. 1997.

[17] T. Kaldewey, C. Lin, and S. A. Brandt. Firm real-time processing in an integrated real-time system. In *Work in Progress Session of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, California, 2006.

[18] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS 1995)*, Dec. 1995.

[19] C. Lin and S. A. Brandt. Improving soft real-time performance through better slack reclaiming. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS 2005)*, pages 3–14, Miami, Florida, Dec. 2005.

[20] J. W. Liu, K. Lin, W. Shih, A. C. Yu, J. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 25(5):58–68, May 1991.

[21] P. Martí, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes. Optimal state feedback based resource allocation for resource-constrained control tasks. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004)*, pages 161–172, Dec. 2004.

[22] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the 1994 IEEE International Conference on Multimedia Computing and Systems (ICMCS '94)*, pages 90–99, May 1994.

[23] J. Nieh and M. Lam. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, Oct. 1997.

[24] L. Niu and G. Quan. A hybrid static/dynamic dvs scheduling for real-time systems with (m,k)-guarantee. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS 2005)*, pages 356–365, Miami, Florida, Dec. 2005.

[25] A. Povzner, C. Lin, and S. A. Brandt. Supporting rate-based processes in an integrated system. In *Work in Progress Session of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, California, 2006.

[26] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS 1997)*, Dec. 1997.

[27] J. Regehr. *Using Hierarchical Scheduling to Support Soft Real-Time Applications on General-Purpose Operating Systems*. PhD thesis, University of Virginia, May 2001.

[28] J. Regehr and J. A. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, pages 3–14, London, UK, Dec. 2001. IEEE.

[29] H. Tokuda and T. Kitayama. Dynamic QoS control based on real-time threads. In *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 114–123, 1993.

[30] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI'94)*, Nov. 1994.