Firm Real-Time Processing in an Integrated Real-Time System

Tim Kaldewey Caixue Lin Scott Brandt
Univ. of California, Santa Cruz
{*kalt, lcx, sbrandt*}*@cs.ucsc.edu*

Abstract

We explore the integration of firm real-time processing, where processing completed after its deadline has no value but some jobs may be terminated or skipped, into an integrated real-time system managing hard, soft, and non-real-time processes. We show that it is feasible to add firm real-time processing to an integrated environment and that concurrently executing soft real-time processes can benefit from the slack made available from dropped firm real-time jobs. We examine how dropping jobs of firm real-time tasks using different static and dynamic drop patterns reduces deadline misses of soft real-time tasks. Our results show that even a simple static drop pattern results in noticeable performance gain for other soft real-time processes and argue that our dynamic approach, once implemented, will produce even better results.

1 Introduction

The boundaries between real-time and best effort computing are becoming increasingly blurred. In the beginning real-time was understood as what refer now to as hard real-time—processes whose tasks are not allowed to miss any deadline. Particularly in the multimedia domain, many applications exist which are not necessarily required to meet all of their deadlines. These soft real-time processes vary broadly in their requirements and many subclasses have been identified [3]. In order to achieve better CPU utilization, schedulers supporting soft real-time processing can let such processes enter the system even though their worst-case execution times cannot be guaranteed. At the same time they need to ensure that in case of high system load hard real-time deadlines are still met and best-effort processes are not starved.

Lin and Brandt showed that in an integrated environment supporing hard, soft, and non-real-time processes, soft real-time processes can benefit greatly from dynamic slack generated by both hard and soft real-time processes [7]. Firm real-time [14], or weakly hard real-time [1] tasks, only need to meet a certain number of deadlines, allowing a certain number of jobs to be skipped. Our goal is to extend our integrated real-time system [3] to natively support firm real-time processing, and to exploit the firm real-time property to increase the amount of available slack—at exactly the right time—to further improve the performance of soft real-time and best-effort processes.

2 Related Work

In the literature firm real-time is viewed as a hybrid category between hard and soft real-time. Firm real-time processes are always required to meet a certain amount of deadlines in a given time window, described by a certain number of consecutive invocations. Different opinions exist about the value and handling of tasks missing deadlines. The approaches range from letting late tasks finish at low priority or aborting their execution, without providing any results to the application.

Traditionally deadline misses for RT processes have been expressed as a maximum allowable loss percentage, also refereed to as Skip-Over Model [8]. This description is insufficient for firm real-time applications because it assumes that deadline misses are adequately spaced. The approach taken by by Hamadaoui and Ramanathan characterizes firm real-time processes by using (m,k)-firm deadlines, defined as meeting m deadlines out of k consecutive task invocations of a process [9]. While this approach was targeting multimedia streams in a Network, Quan and Hu [12] explicitly tackle CPU scheduling as an application domain for (m,k)-firm constraints. A similar approach was taken In a series of three papers by Burns and various colleagues from the University of York [1, 5, 2][1]. These approaches continue processing of late results at low rates, while others acknowledge the fact that late results have no value.

In their paper on paper on QoS functionalities for the Linux real-time Application Interface, Marchand and

[1] They prefer the term Weakly Hard Real-Time

Silly-Chetto describe a dynamic slack stealing algorithm based on Earliest Deadline Latest scheduler [10]. It discards selected task instances without violating firm real-time constraints set by the application. The focus is on optimizing the response time of aperiodic tasks in the presence of firm real-time workloads. An approach with the same intent has been undertaken by Thomas et al. [13] pointing out that in general the major part of available slack is non-uniformly distributed. Niu and Quan [11] provide a proof that in static drop patterns uniformly distributed slack (created by dropped firm real-time tasks) provides optimal performance for DVS systems. Xiu et al. [14] use Markov Chains to characterize the behavior of firm real-time processes which allows task models different from the previously used periodic/aperiodic ones. Their approach is intended for use in networked feedback control systems where the result of one task execution influences the next. Here also tasks are rather dropped than results delivered late.

3 Integrating firm real-time into an integrated environment

We follow the newer approaches [10, 14, 11, 13], that late results do not have any value for a firm real-time applications. This implies that firm real-time tasks are not scheduled if they cannot meet their deadlines and terminated at their deadline if they have not finished by then. Such a behavior can be illustrated using a process decoding a video stream in a fully loaded system. When video frames are delivered late the playback is perceived as choppy. Skipping frames and ensuring the next frames are delivered on time may be less disturbing and might not be noticeable for the spectator, depending on the content of the skipped scene. One might argue that contemporary personal computers need only a fraction of their CPU time for decoding a video, but in case of interrupts caused by mixed workloads continuous playback might not be feasible. Considering small mobile devices with a fraction of the processing power of a PC this approach becomes more sensible; video on mobile phones is becoming very popular in Europe. Giving the resources to a video process as if it were a hard real-time task might not leave resources for other functionalities or might not be possible at all because of resource limitations.

The RAD model [3] characterizes real-time processes by their resource needs (how much?) in relation to their dispatching requirements (when?). It distinguishes hard real-time, best-effort and soft real-time processes, the latter subdivided into missed-deadline, resource-adaptive, and rate-based soft real-time. In this model, firm real-time is a hybrid category between hard real-time and resource-adaptive soft real-time (Figure 3). A firm real-time process can lower its resource consumption by dropping jobs, but the minimum service has to be guaranteed. Therefore firm real-time workloads can be describes as consisting of two parts: a hard real-time part representing the process minimum requirements and a resource-adaptive part. How to support these tasks and optimally exploit the resource adaptive part for performance improvements of other tasks is the subject of our current research.

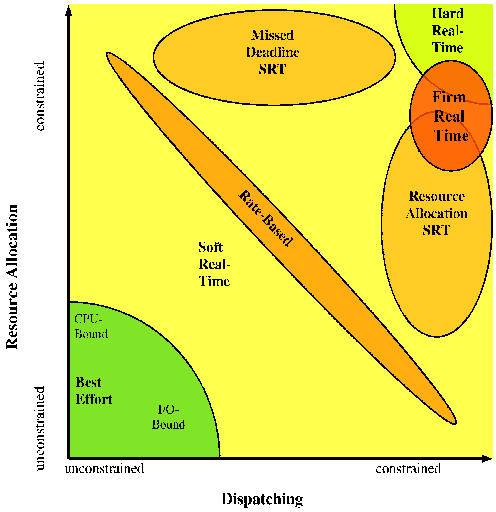
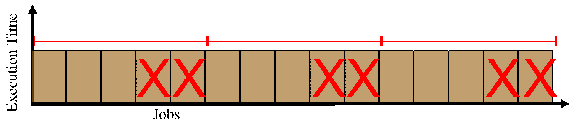


Figure 1: The RAD model extended with a process class for firm real-time

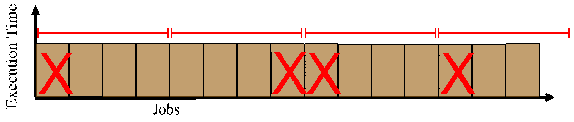
After choosing to skip permissible jobs of firm real-time tasks in favor to soft real-time tasks in cases of high system load, the question is which ones to skip. We evaluate the gain of compromising firm real-time tasks for soft real-time performance in a dynamic integrated system. We use the (m,k) model to discuss static and dynamic approaches for dropping firm real-time jobs searching for an optimal approach. Preliminary results confirm that static patterns work well. We are investigating dynamic dropping strategies expecting them to deliver superior results.

3.1 Exploiting the characteristics of firm real-time

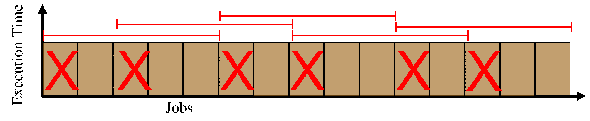
After deciding to drop jobs of firm real-time tasks the difficulty is to do it in way that a) the minimum requirements are still met and b) that other processes can make optimal use of the generated slack. Different static approaches (drop patterns) were examined by Koren and Shasha [8] and Niu and Quan [11]. In the first always k consecutive executions form a static window, during which always the



(a) a static drop pattern (deep Red)



(b) transformed to $(3,4)$ pattern



(c) dynamic drop pattern

Figure 2: Firm real-time process with $(3,5)$ constraints using different drop patterns

same m tasks are executed, in the deep red pattern it is always the first ones. It is obvious that the drop pattern can be chosen arbitrarily, since the pattern is chosen once for all windows (Figure 2(a)).

Niu and Quan [11] prove that among the static patterns a uniform distribution for dropped tasks provides optimal performance. As shown in Figure 2(b) an (m,k)-firm constraint can be transformed into a $(k-1,k)$ one, by choosing an appropriate k. While choosing the new k equal to $m+1$ is always feasible, better choices for k may exist. We are looking for a formal approach to find the optimal value for k.

Since static drop patterns provide no flexibility regarding when and if to drop jobs, we expect better results from a dynamic approach which drops jobs on demand. In order to be able to drop firm real-time jobs dynamically without violating the tasks' constraints a sliding window mechanism needs to be applied. The window cannot be kept static if dropped tasks are chosen dynamically, otherwise the constraints described by (m,k) could be violated (Figure 3.1). A sliding window mechanism keeping track of the last $k-1$ executions allows the system to determine whether the current task can be dropped without violating the given constraints (Figure 2(c)). Considering dropping tasks only when needed avoids the problem of static drop patterns also encountered by Butazzo and Caccamo [6] that spare time generated by dropped tasks has a "granular" distribution and cannot be reclaimed at any time, hence it may render the CPU idle. This is not desirable, especially when other tasks are missing their deadlines.

Static drop patterns will influence the Quality of Service received by firm real-time task permanently. Dynamic drop patterns ensure that firm real-time tasks

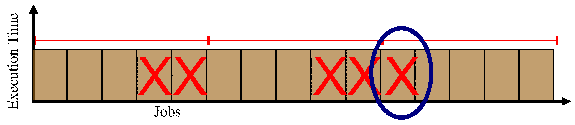


Figure 3: Firm real-time process violating its $(3,5)$ constraints

achieve maximum performance when no other tasks need additional CPU time, not wasting CPU time by unnecessarily dropping tasks.

3.2 Preliminary Implementation

We extended the BACKSLASH slack scheduling algorithm [7] in the RBED framework for the Linux 2.6.8.1 Kernel with firm real-time capabilities. For validation, we first implemented the simplest static pattern, deeply Red, described in Section 3.1. We plan to implement the other static dropping patterns and the dynamic drop mechanism in our future work. Since we consider firm real-time as a subclass of soft real-time, we extended the soft real-time specification with a subtype for firm real-time adding parameters for m and k. We use the firm real-time semantics described in Section 3, in case of high system load only scheduling the firm real-time tasks needed to fulfill the basic requirements.

Firm real-time processing requires that applications properly handle dropped jobs. Even though this may sound like a fundamental change, in current applications this is not the case. Video applications in the early days of personal computers introduced the concept of skipping frames(tasks) to prevent the output from looking choppy on slower machines. Using our scheduler, an application receiving the result of a non-consecutive job (or frame) can assume that the execution of the intermediate ones has been skipped and continue without waiting for them.

In order to simulate this behavior with a workload generator, we implement a system call to query a flag, indicating whether the current task is going to be executed or not. In case the flag indicates a task drop the process sleeps until the next task release without consuming further CPU resources. This corresponds to a task drop by the scheduler in case of fully firm real-time capable applications.

3.3 Preliminary results

Our sample workload includes 2 hard real-time processes accounting for 30% of the workload, a firm real-time process with 28% and (m,k) constraints of $(5,3)$ and one soft real-time process with 50%, detailed in Table 1. We reserve 2% of CPU utilization for the best-effort processes.

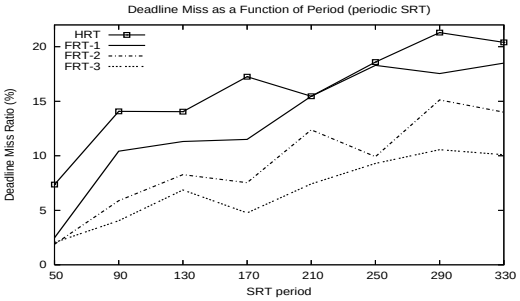


Figure 4: Firm real-time dropping pattern impact on soft real-time performance

Table 1: sample workload to evaluate firm real-time (unit in *ms*)

process type	CPU utilization	wcet	period
HRT	10%	20	200
HRT	10%	30	300
FRT	28%	28	100
SRT	50%	25-165	50-310

We compare the performance of soft real-time processes in the presence of a firm real-time process conventionally scheduled as hard real-time to scheduling it as firm real-time with varying parameters for $(5,3)$. Since several sources [6, 13, 11] indicate that the distribution of the slack and the consuming tasks influences reclaimable slack we also vary the period length of the soft real-time task while keeping the workload constant by changing the execution time accordingly.

Figure 3.3 shows the deadline miss ratio of the soft real-time task as a function of its period. The first curve (HRT) in the figure represents the firm real-time scheduled as a hard real-time without dropping jobs; the other three curves (FRT-*n*) represent respectively the case that *n* jobs of the firm real-time task will be dropped consecutively in every $m = 5$ jobs.

As we can see, compared to non-drop mechanism (HRT), the static drop pattern always results in equal or better soft real-time deadline miss ratio for the sample workload.

4 Conclusion and Future work

In this paper we describe the characteristics of firm real-time processes and discuss several approaches to exploiting them in order to generate slack for other process classes, in particular soft real-time processes. We provide an implementation as an extension for the RBED scheduler for Linux using a simple deeply Red drop pattern. The results of this first prototype, greatly improving soft real-time performance, encourages us in to explore the potential optimized static and our dynamic drop patterns. Our future work will include implementing and testing the performance of the remaining approaches for task dropping and proving that exploiting firm real-time characteristics always results in better or equal performance for soft real-time tasks. Even though we can improve soft real-time performance by dropping firm real-time tasks, the question if it is worth it to improve soft real-time performance has to be answered. Therefore future work will also include approaching this question along the lines of the Dynamic Quality of Service management (DQM) approach previously examined by Brandt et al. [4].

References

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

[2] G. Bernat and R. Cayssials. Guaranteed on-line weakly hard real-time systems. *Proceedings. 22nd IEEE International Real-Time Systems Symposium*, pages 25–37, December 2001.

[3] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. *Proceedings. 24th IEEE International Real-Time Systems Symposium*, pages 396–407, December 2003.

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

[5] I. Broster, G. Bernat, and A. Burns. Weakly hard real-time constraints on controller area network. *Proceedings. 14 th Euromicro Conference on Real-Time Systems*, pages 134–141, 2002.

[6] G. Buttazzo and M. Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Transactions on Software Engineering*, 25(1):22–32, Jan-Feb 1999.

[7] C.Lin and S. Brandt. Improving soft real-time performance through better slack reclaiming. *Proceedings. 26th IEEE International Real-Time Systems Symposium*, pages 3–14, December 2005.

[8] G.Koren and D. Shasha. Skip-over: algorithms and complexity for overloaded systems that allow skips. *Proceedings. 16th IEEE International Real-Time Systems Symposium*, pages 110–119, December 1995.

[9] 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, December 1995.

[10] A. Marchand and M. Silly-Chetto. Qos and aperiodic tasks scheduling for real-time linux applications. *6th Real Time Linux Workshop*, November 2004.

[11] L. Niu and G. Quan. A hybrid static/dynamic dvs scheduling for real-time systems with (m,k)-guarantee. *Proceedings. 26th IEEE International Real-Time Systems Symposium*, pages 356–365, December 2005.

[12] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. *Proceedings. 21st IEEE International Real-Time Systems Symposium*, pages 79–88, December 2000.

[13] D. Thomas, S. Gopalakrishnan, M. Caccamo, and C. Lee. Spare cash: Reclaiming holes to minimize aperiodic response times in a firm real-time environment. *Proceedings,. 17th IEEE Euromicro Conference on Real-Time Systems*, pages 147–156, July 2005.

[14] D. Xiu, X. Hu, M. Lemmon, and Q. Ling. Firm real-time system scheduling based on a novel qos constraint. *Proceedings. 24th IEEE International Real-Time Systems Symposium*, pages 386 – 395, December 2003.