

Efficient Soft Real-Time Processing in an Integrated System

Caixue Lin and Scott A. Brandt
University of California, Santa Cruz
{lcx,sbrandt}@cs.ucsc.edu

Abstract

The rapidly increasing user demand for more powerful computing platforms and application requires modern operating systems capable of simultaneously supporting processes with a variety of different timeliness constraints. It requires real-time applications to guarantee their timeliness constraints by using worst case resource reservation. Non-critical applications (such as soft real-time) do not reserve worst-case resources, and may therefore execute with degraded performance. Reservation-based mechanisms may overbook resources so that the system wastes large amounts of resources in such a way that the unreserved reservations are not efficiently consumed. In this paper, we present an integrated system with a flexible and efficient resource management mechanism for soft real-time processes so that slack time is better utilized. The simulation results show that our mechanism results in significantly performance improvement in terms of reducing the deadline miss ratio and tardiness for soft real-time processes.

1 Introduction

The rapidly increasing user demand for more powerful computing platforms and application requires modern operating systems to support scheduling of processes with a variety of different timeliness constraints in an integrated way. Example works are the hierarchical scheduler developed by Regehr *et al.* [9] and the flat integrated scheduler RBED by Brandt *et al.* [3]. In such systems, worst case resource reservation or execution is usually used to guarantee the performance of critical real-time applications, such as external event/signal sampling and processing; non-critical applications, including soft real-time (*e.g.* desktop audio/video) and best effort (*e.g.* compiler), receive the left-over resources. However the constant worst case reservation mechanism always overbooks the resources that a real-time application does not really need, so slack time is generated when a process' job completes before its reservation is consumed. As a result, the system may waste resources.

In this paper, we present a slack reclamation mechanism that improves the performance of soft real-time applications and still guarantees the worst case reservation of hard real-time applications. Our preliminary results show that this

flexible scheduling mechanism results in significant performance improvement for soft real-time processes in terms of reducing their deadline miss ratio and tardiness compared to the well-known CBS [1] mechanism and the BEBS algorithm [2].

2 Related work

A traditional slack stealing algorithm [6] schedules aperiodic jobs whenever the periodic tasks and sporadic jobs have slack, that is, their execution time can be safely postponed without causing them to miss their deadlines. Its main drawback is the incurred overhead to compute the amount of the available slack at each scheduling decision time.

CBS [1] provides a greedy slack time reclamation mechanism by immediately releasing the next job of an expired process with the deadline set to the end of its next period. Slack time is automatically used by acyclic or cyclic processes that allow the system to re-phase¹ their jobs' release times. Our work aims at utilizing the slack time for periodic soft real-time processes that do not allow re-phasing their jobs' release time. GRUB [5] is a CBS-alike algorithm. It tends to dynamically allocate excess capacity to the current running server or a few active servers in direct proportion to their processor shares. It has to frequently decide the timing and duration slack time (idle time) is available. Furthermore the available slack time is dynamically re-allocated to the needy servers by updating their reservations. These dynamic operations incur a large overhead.

The IRIS principle [7] enhances CBS with a fairer slack reclaiming strategy and guarantees a minimum execution to a task in a fixed interval of time. Similar to IRIS, BEBS [2] models best-effort applications as aperiodic tasks. In both IRIS and BEBS, slack time is pushed back to the end of a point only until which it is available to be used fairly. Our work focuses on improving the performance of soft real-time processes in terms of tardiness and deadline miss ratio by consuming the slack time as early (and not fairly) as possible.

¹Note that to re-phase a job's release time means to dynamically adjust its pre-defined release time to a new one so that the job can be released earlier or later than the pre-defined one.

Our previous work RBED [3] provides a CBS-like slack time management mechanism in which only best-effort processes consume the slack left by hard real-time and soft real-time processes. It does so by immediately releasing the next job of an expired best-effort process with the deadline set to the end of its next period. With this technique, though slack time is evenly distributed among the runnable best-effort processes, a newly-entered best-effort task may cause others to starve; also it does not allow other classes of processes (notably soft real-time) to take advantage of dynamic slack.

3 System model and problem description

3.1 System model

The system we are considering is an integrated system (as described in our previous work RBED [3]) consisting of hard real-time, soft real-time and best-effort processes. Our system uses separate rate-based resource allocation and EDF scheduling algorithms for process management.

3.1.1 Rate-based resource allocation

The rate-based resource allocation mechanism, which is similar to the reservation-based resource allocation (such as Processor Capacity Reserves [8]), allocates a fixed minimum resource budget to each process in a fixed interval.

In our system, each periodic real-time (hard and soft) process T_i consists of sequential jobs $J_{i,k}$, where each job has a release time $r_{i,k}$, fixed period p_i , deadline $d_{i,k}$, and fixed minimum execution budget e_i . The resource rate (utilization) of T_i is $u_i = \frac{e_i}{p_i}$. All best-efforts processes are managed by a single pseudo periodic server (called BEServer) which has a pseudo period $p_{beserver}$ and a pseudo execution budget $e_{beserver}$ (The detailed management conducted by the BEServer is explained in [4]). Note that the release time of a periodic process (except the BEServer) in our system can not be re-phased dynamically during scheduling. The total global system utilization is $U_g = U_{HRT} + U_{SRT} + u_{beserver}$, where $U_X = \sum_{T \in X} T.u$ and X is the set of all tasks of type X . Specifically, upon request at process creation time, the rate-based resource allocation mechanism determines the execution budget e_i for process T_i according to the following rules subject to $U_g \leq 1$:

- If T_i is hard real-time process, e_i is assigned to its worst case execution time so that all its deadlines are guaranteed to be met. T_i is rejected if $U_g > 1$ upon the allocation.
- If T_i is soft real-time process, e_i is assigned to an allocated bandwidth, which is usually less than the worst case but more than the average case. T_i is rejected if $U_g > 1$ upon the allocation.
- For the BEServer, a minimum resource utilization β is reserved for it to guarantee no best-effort processes is starved, *i.e.*, $u_{beserver} = \max(\beta, 1 - U_{HRT} - U_{SRT})$.

3.1.2 EDF scheduling and one-shot mechanism

Our proposed mechanism uses EDF to schedule processes with resource reservation and overrun protection enforced by a high-precision one-shot timer. Since U_g never exceeds 1, the EDF scheduling algorithm always provides the required guarantees to the different classes of processes in the system (see [4] for details).

Note that process overrun is protected by issuing a one-shot timer to the current running process with the timer counter set to the process' left budget c_i (c_i of a process is initialized to its assigned budget e_i). When the one-shot timer expires, processes are rescheduled as a normal pre-emption happens. Here, we only consider soft real-time process overrun since it uses average execution reservation. We assume a job is never dropped in such a way that an overrun soft real-time process may still re-start the overrun part of a job upon its next release (Note that an alternative is to notify the application to drop its overrun job if drop is allowed). An example of process overrun is given in Figure 1(a), which shows the CPU allocation for three real-time (soft or hard) processes $T1$, $T2$ and $T3$ in the system with following configurations respectively: $(p_1 = 6, e_1 = 1.5)$, $(p_2 = 8, e_2 = 4)$ and $(p_3 = 10, e_3 = 2.5)$. Assume the first job of $T1$ has actual execution time 2, which is larger than its budget $e_1 = 1.5$. Thus its first job overruns and the overrun part is pushed back until the next release time.

3.2 Problem description

There are drawbacks when using the simple EDF scheduling algorithm with the one-shot mechanism for an integrated system in which hard real-time and soft real-time processes may generate slack time at any time because of highly varying execution times but constant reservation. Figure 1 shows the problems. First, slack time may be not used by a past overrun job of a process. In Figure 1(a), the first job of $T1$ has an overrun part of 0.5; the first job of $T2$ has actual execution time of 2 and thus it generates slack time of 2. Since there is no slack time management employed, the overrun job of $T1$ can't start to execute until time 6 and thus misses its deadline. Second, slack time may be not used by a future (following) overrun job of a process. In Figure 1(b), the first job of $T1$ has actual execution time of 1 and thus it generates slack time of 0.5; the first job of $T2$ has an overrun part of 0.5. Again without slack time management, the generated slack time is pushed back to a far idle point until which it is available. As a result, the overrun job of $T2$ misses its deadline.

4 Efficient slack time management

The basic principle for our slack time management is to use slack time as early as possible for the soft real-time processes that need it most. Note that our performance metrics for soft real-time processes are tardiness and deadline

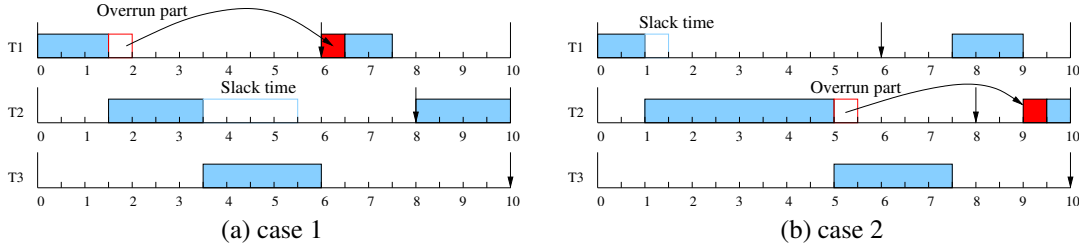


Figure 1. Drawbacks of EDF scheduler with one-shot support

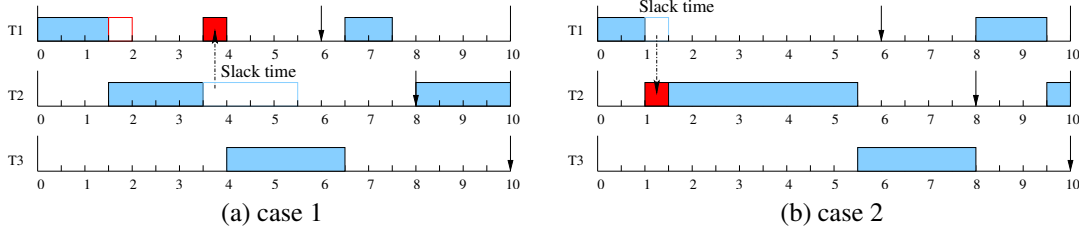


Figure 2. Slack time management for soft real-time processes

miss ratio, but not fairness in using slack time. As described in section 3.1.2, the currently running process is associated with a one-shot timer that controls the process' maximum execution. The one-shot timer starts or updates its counter value according to the process' leftover budget when the process starts or resumes to run.

When the one-shot timer expires before the running process completes, a predefined timer handler is called to set the status of the currently running process to *expired* and trigger process re-schedule. In re-schedule, the currently expired process is preempted and at the same time the EDF scheduler selects the next runnable process with the earliest deadline in the run queue.

When the currently running process completes or blocks before the one-shot timer expires, slack time is generated. Upon process completion or blocking, re-schedule is again triggered. In this case, the expired process with the earliest deadline is selected to run with the remaining one-shot timer (see Figure 2(a) for example). This is done by resetting the status of the selected expired process to *runnable* and artificially creating for it a virtual deadline equal to the deadline of the process which generates slack time. Note that the actual deadline (*i.e.*, the next release time) of the selected process is kept unchanged because its next job release can't be re-phased. During this whole process, no dynamic computation or resource re-allocation is needed. In this way, the slack time is allowed to be consumed by the expired processes using EDF until the timer expires. In case there is no expired process available, the next runnable real-time (hard or soft) process with the earliest deadline is instead selected to run with the remaining one-shot timer. Note that the BEServer is an exception, in which the slack time is transferred to the future runnable processes until the one-shot timer expires (see [4] for details).

The run queue is sorted by earliest deadline first. We use

two list heads, expired-head and runnable-head to respectively index the first expired process and the first runnable process with the earliest deadline in the run queue. Thus no overhead is incurred to pick a process (either expired or runnable) with the earliest deadline to run.

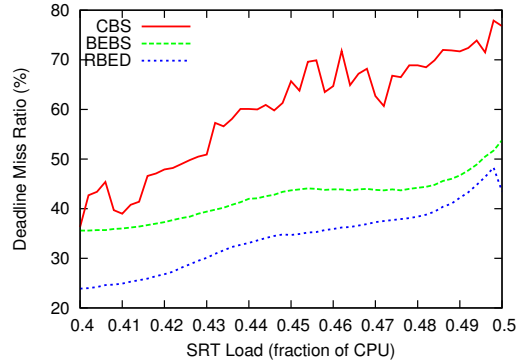
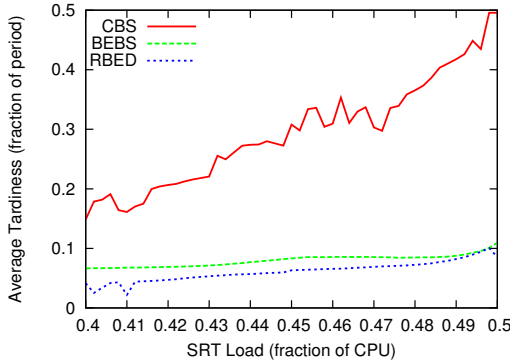
Our slack time solution solves the problems presented in Figure 1. The corresponding resulting schedules are showed in Figure 2. Clearly, in both examples, with slack time management, no job misses their deadlines even though there are overrun jobs. The advantages of using EDF scheduler with one-shot mechanism are summarized as follows:

- Slack time is always consumed as early as possible by processes that need it most. Thus the tardiness and deadline miss ratio of soft real-time processes are reduced.
- EDF is used as a universal scheduling algorithm both in normal process scheduling and slack time management.
- Scheduling overhead is low since no dynamic complex computation on scheduling parameters are required.

In our system, in order to reduce the overhead caused by dynamic schedule, all best-effort processes are in turn selected to run by a BEServer, which is a pseudo soft real-time process scheduled by the EDF. The BEServer is treated as a normal soft real-time process, with the exception that BEServer is not allowed to use slack time produced by other real-time processes until there are no expired or runnable real-time processes in the system. For space limitation, we do not show the detailed design of the BEServer and its advantages (refer to [4] for more details).

5 Preliminary results

We simulated our slack time management technique and compared its performance to CBS as well as BEBS. Note that in both CBS and BEBS, we artificially allow the jobs'



(a) Tardiness as a function of load ($p = 40\text{ms}$)

(b) Deadline Miss Ratio as a function of load ($p = 40\text{ms}$)

Figure 3. SRT performance comparison among CBS, BEBS and RBED

release times of a process to be re-phased so that comparison can be conducted. For simplicity, we call our system using EDF with one-shot mechanism as RBED.

One of our simulation workloads consists of five hard real-time (HRT) tasks and one soft real-time (SRT) task. All the HRT tasks have the same scheduling parameters ($p = 40\text{ms}$, $e = 4\text{ms}$) and thus each of them takes 10% of the CPU usage; the SRT task has fixed period $p = 40\text{ms}$, but varying average reservation by increasing its average usage from 40% (corresponding to $e = 16\text{ms}$) to 50% (corresponding to $e = 20\text{ms}$) during the simulations. The actual execution times for each job of the five HRT tasks and the SRT task are generated randomly in such a way that the actual execution time of each HRT task is never larger than its reservation $e = 4\text{ms}$ and the average actual execution time of the SRT task is more or less equal to its reservation e . Figure 3 shows the performance comparison in terms of average tardiness (fraction of the SRT period) and deadline miss ratio for the SRT task in CBS, BEBS and RBED. Clearly, RBED outperforms CBS and BEBS regardless of the load of the SRT task. For space limitation, we do not show the other results from our simulations with multiple SRT tasks (please refer to [4] for more detail), but the results are similar. We also recorded the context switch number for each run in our simulations. The result is by using our slack time management no extra overhead is introduced compared to CBS or BEBS.

6 Conclusion and future work

We present a slack time management technique, which allows soft real-time processes to use generated slack time as early as possible. Our simulation results show significant performance improvement for soft real-time processes in terms of smaller tardiness and lower deadline miss ratio by using this technique other than CBS or BEBS.

Our future work is to implement the efficient soft real-time resource management technique (including slack time management and best-effort management) in a real system,

such as Linux 2.6, and thoroughly investigate its performance by performing extensive experiments.

Acknowledgments This research was funded in part by Intel Corporation. We gratefully acknowledge comments and suggestions by Scott Banachowski, Pau Martí and the review committee.

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] S. Banachowski, T. Bisson, and S. A. Brandt. Integrating best-effort scheduling into a real-time system. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004)*, Dec. 2004. To appear.
- [3] 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.
- [4] C. Lin and S. A. Brandt. Efficient soft real-time processing in an integrated system. Technical Report UCSC-CRL-04-11, University of California, Santa Cruz, Sept. 2004.
- [5] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 193–200, June 2000.
- [6] J. W. Liu. *Real-Time Systems*. Prentice-Hall, 2000.
- [7] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. IRIS: A new reclaiming algorithm for server-based real-time systems. In *10th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS04)*, May 2004.
- [8] 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.
- [9] 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.