Improving Control Performance Using Discrete Quality of Service Levels in a Real-Time System †

Caixue Lin, Pau Martí, Scott A. Brandt, Scott Banachowski
Computer Science Department
University of California
Santa Cruz, CA, USA
{lcx,pmarti,sbrandt,sbanacho}@cs.ucsc.edu

Manel Velasco and Josep M. Fuertes
Automatic Control Department
Universitat Politècnica de Catalunya
Barcelona, Spain
{manel.velasco,josep.m.fuertes}@upc.es

Abstract

Traditional control systems employ fixed sampling intervals. Recent work in integrated control and real-time systems has resulted in control systems in which the sampling interval may vary based on the state of controller performance. Modern real-time systems provide mechanisms for dynamically adapting application resource usage based on system state and application needs. In this paper we investigate employing this mechanism to allow several control applications to dynamically adapt their resource usage so that they receive enough of the limited resources to achieve their goals, but do not greedily consume resources, allowing the system to be utilized by other applications as well. Our preliminary results show that this technique can result in significantly lower controller error (by an average of over 20% in our experiments) with no increase in overall resource usage.

1. Introduction

Recent work in flexible real-time provides applications with the opportunity to adapt to available system resources [2]. When processes do not meet enough deadlines to provide adequate Quality of Service (QoS), they adapt by changing their processing requirements. Similarly, researchers have examined ways to provide similar soft functionality to control applications by allowing the applications to change their sampling intervals, along with their control law, on the fly [4]. The reasoning is that when the system is stable, the control application need not monitor and control the plant as aggressively, and by reducing the sampling interval, the task consumes less resources than the worse-case. Generally, smaller sampling intervals consume more resources and yield better control, while larger sampling intervals consume less resources but may yield larger errors.

This paper examines using an adaptive Quality of Service (QoS) framework to support these new flexible control applications. The contributions of this work are the implementation of an adaptive QoS soft real-time scheduling framework in an integrated real-time scheduler, and the application of this framework to the problem of supporting flexible control applications. The novel aspects of the design include the elimination of static benefit functions associated with different levels of QoS in favor of dynamic benefits based on a novel function of instantaneous controller error. Overall, we show scenarios in which this technique reduces the error of an individual controller by as much as 40% and reduces the average error of all executing controllers by about 22%.

2. Flexible Real-Time Processing

We implemented the Adaptive QoS system in the Rate-Based Earliest Deadline (RBED) Linux system [2].

† This work was supported in part by Intel Incorporation, by Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya, and by Spanish Ministerio de Ciencia y Tecnologia Project ref. DPI2002-01621.

RBED provides fully integrated scheduling of hard real-time, soft real-time, and best-effort processes. RBED dynamically adjusts both the utilizations and periods of applications so that it flexibly supports several flavors of real-time, soft real-time and best-effort processing.

To support Adaptive QoS applications, we have implemented a modified version of the DQM QoS Level real-time system [1] in the RBED system. QoS Levels allow discrete application adaptation. Each application provides to the system a table specifying the discrete *levels* at which it can operate, the relative amount of resources required to run at each level, and the relative *benefit* of running at each level. Then, a QoS manager analyzes the information provided by each application to determine its allocation strategy.

3. Flexible Control

Traditional control specifications are used to obtain static controllers; given a set of different control application scenarios, a controller is designed with enough robustness to cope with all of them while achieving the desired application performance. When using a static controller we obtain the same average performance (e.g., benefit) for all application scenarios. However, if we execute specific controllers for each application scenario, we may maximize the overall benefit by executing each controller according to the application dynamics.

Consequently, in our system we design different controllers for different scenarios, each one delivering a specific performance level (benefit) and demanding a specific CPU share (utilization). Note that each controller's benefit relates to control performance in the sense that higher execution rates yield better control performance. Therefore, it is desirable to execute each task with the highest rate controller. However, this is not always feasible due to the limited availability of computing resources. In a situation with several tasks, each one with few candidate controllers, we must choose for each task the appropriate controller such that the overall benefit is maximized taking into account resource availability. To solve the problem of choosing the appropriate combination of controller levels, we allow the system to obtain feedback information from the controlled plants. To do so, the norm of the state vector of each controlled process (that may be appropriately weighted), which captures all of the control system dynamics, will be used as a feedback information. Based on the feedback information (i.e. the control errors) from the control tasks, the system re-scales the benefits, and from these dynamic benefit values the best controller per task is chosen (refer details to [**?**]).

It is worth to mention that in Cervin et al. [3] a system in which feedback from control tasks is used to adjust the workload by rescaling the task periods is presented. However, in this work all periods are rescaled each time, and there is no provision to trade-off resources among tasks that need them more urgently.

4. Results

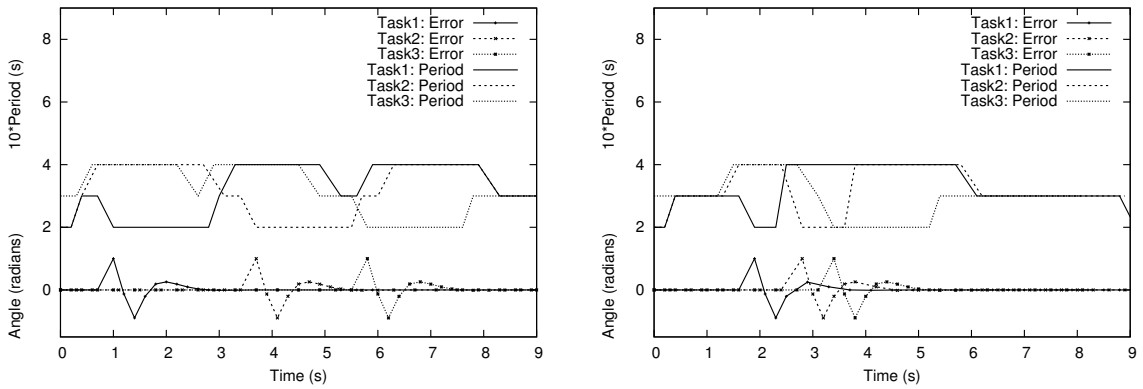
To show the performance achieved using dynamic period adjustment, we simulated the control tasks, and ran them in the RBED scheduler. The simulated control tasks were configured with the QoS Levels described in Table 1.

Table 1: Benefit tables for the control tasks

Number of QoS Levels: 4			
Level	Benefit	% CPU	Period
1	0.80	40%	0.2 s
2	0.56	27%	0.3 s
3	0.40	20%	0.4 s
4	0.32	16%	0.5 s

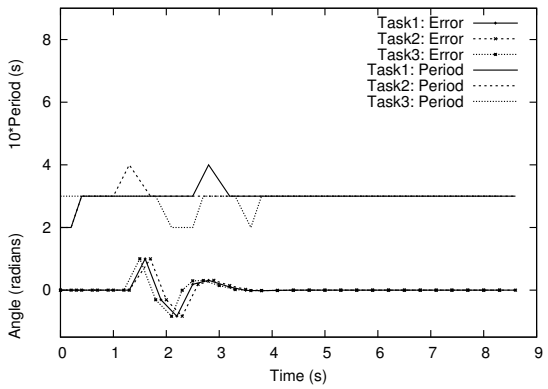
We evaluate the performance of the system by comparing the performance of control tasks controlling inverted pendulums with and without dynamic scaled adaptation. In the non-flexible case, benefits are static so the period chosen by the controller does not change throughout the execution. With adaptation, the benefit of each QoS level is scaled in proportion to the dynamic control error. This means as the controller executes its benefit will be scaled, possibly triggering the QoS adaption mechanism to alter the sampling periods of the controller tasks.

To evaluate the performance of the system, we consider three control tasks. A hard real-time task HRT with

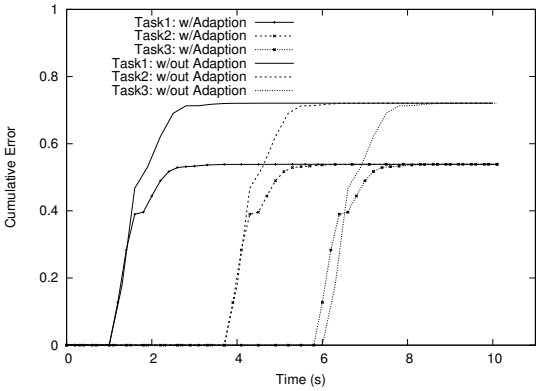


(a) Adaptation (not overlapped)

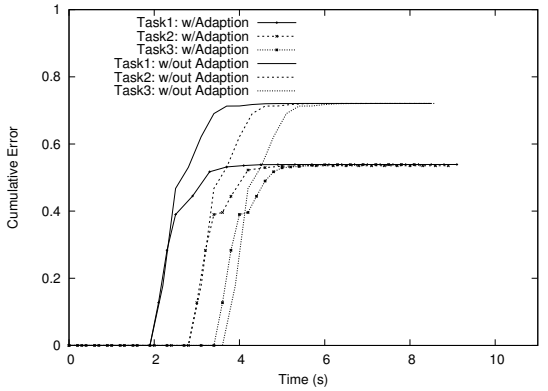
(b) Adaptation (partially overlapped)



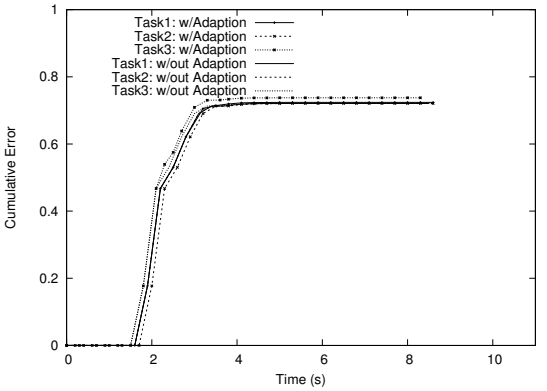
(c) Adaptation (fully overlapped)



(d) Cumulative error (not overlapped)



(e) Cumulative error (partially overlapped)



(f) Cumulative error (fully overlapped)

Figure 1: Summary of Results. Graphs (a), (b) and (c) show the variation on control tasks periods according to the inverted pendulum error, for each scenario. Graphs (e), (f) and (g) show the cumulative error achieved by each control task with and without adaptation, for each scenario.

period of 0.5s and execution time of 0.05 s. (10% CPU bandwidth) and a CPU bound best-effort task BE are also running simultaneously with the control tasks. The control tasks start running at the same time and initially are in stable states. We perturb each task controlled system at different times in the different scenarios. First, a perturbation is triggered at different times for each control task with large enough gaps that only one task has controller error at any one time, i.e., not overlapped. Second, a perturbation is triggered at different times for each control task with small enough gaps that multiple controllers have error at the same time, i.e., partially

overlapped. Third, a perturbation is triggered simultaneously for all control tasks, i.e., fully overlapped.

Figures 1 (a), (b) and (c) show the resulting performance of the control tasks in these scenarios. In Figure 1(a) the system dynamically reallocates the system resources to allow the controller with the greatest error to run at a higher sampling frequency while lowering the sampling frequency of the controller with the least error. As the perturbations are sufficiently far apart, there is only one application with any error at a time and this application is always allowed to run at it's highest level while it is responding to the perturbation. This results in an overall controller error of 1.61, 25.3% less than without adaptation (refer details to [**?**]). The results for the case of partially overlapped (Figure 1, (b)) are similar to the previous one except that the partial overlap of the errors requires greater sharing of the resources. Nevertheless, the controller error numbers are almost identical, with total error of 2.16 without adaptation and 1.61 with adaptation for a 25.3% reduction overall. In the last scenario (Figure 1 (c)), the overlap of the errors is nearly complete and no benefit is achieved by reallocating the resources. Thus each controller continues to run mostly at its original level and there is little difference between the two cases.

The accumulated error for each task, shown in Figure 1 (e), (f) and (g), provides a good way to compare the performance of the system in the various scenarios, without and with adaptation. In Scenario 1 (not overlapped) and Scenario 2 (partially overlapped), the tasks with adaptation have much less cumulative error than those without adaptation. In Scenario 3 (fully overlapped), there is almost no gap between the accumulated error of the two cases, again reflecting the lack of room for adaptation due to the overlapping errors.

Overall in these three scenarios the controllers achieved 22.3% less error. While the actual benefit in real system will vary, the artificially constructed worst-case parameters of Scenario 3 (fully overlapped) are unlikely to occur in practice and we expect that the system will therefore provide significant performance improvements in most real situations.

5. Conclusion

Traditional control and hard real-time systems have evolved hand-in-hand. Our work continues this evolution by merging flexible control with adaptive soft real-time processing. We have extended the RBED integrated real-time scheduler to include dynamic QoS Level soft real-time processing, and implemented an flexible control system on this platform. We show significant performance improvement when using our QoS level framework to execute flexible control applications with up to 40% improvement (refer details to [**?**]) in some controllers and an average of 22% improvement over all of our experiments, with no corresponding increase in resource usage (refer details to [**?**]).

References

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

[2] 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.

[3] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23:25–53, 2002.

[4] P. Marti, G. Fohler, K. Ramamritham, and J. M. Fuertes. Improving quality-of-control using flexible time constraints: Metric and scheduling issues. In *Proceedings of the 23nd IEEE Real-Time Systems Symposium (RTSS 2002)*, Dec. 2002.