Experimental Analysis of Bandwidth Reservation Adaptive Schemes for Real-Time Systems*

Augusto Born de Oliveira[1], **George Lima**[2], **Eduardo Camponogara**[1]

[1]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis, SC – Brazil

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
40170-110 – Salvador, BA – Brazil

{augustob,camponog}@das.ufsc.br, gmlima@ufba.br

Abstract. *Adaptive schemes for bandwidth reservation Real-Time schedulers have been the focus of various papers in recent years. Two approaches of note are Feedback Scheduling and Optimization-Based reconfiguration schemes. Through simulation, this paper shows some of the issues, benefits and opportunities for composition of these two approaches.*

1. Introduction

Due to the increasing complexity of Real-Time applications, support for run-time scheduler adaptation has become a major topic in the area of Real-Time Systems. Video encoder and decoder tasks, for example, may present severe variations in computation times between frames, and the encoding/decoding streams of different resolutions may lead to very different utilization patterns.

The problems created by highly variable computation times and periods may be dealt with through the use of reservation based schedulers, which are special approaches that provide temporal isolation such as the Constant Bandwidth Server (CBS) [2]. If tasks are treated by a CBS, it is guaranteed that their utilization will never surpass the value set by the system designer even if the task overruns. Additionally, the dynamic nature of the environment with which Real-Time Systems interact requires dramatic and unpredictable changes in processor allocation between tasks. To support these changes in processor allocation, additional scheduling infrastructure is needed.

The problem of dynamic reconfiguration of Real-Time Systems has been addressed in various forms by other works [4, 5, 8, 9, 11, 16], and two approaches are of note: Feedback Scheduling and Optimization-Based Adaptation. Feedback Scheduling uses feedback control theory to actuate on scheduling parameters, trying to minimize scheduling error. Optimization-Based Adaptation uses optimization algorithms to distribute processor shares between tasks, maximizing the overall system benefit.

In this paper, some of the strengths and weaknesses of these two approaches are explored through simulation. Our goal is to put these approaches into perspective. They are analyzed both in isolation and in conjunction. We illustrate via simulation that by combining both approaches into one integrated infrastructure one can benefit from their

*This research has been supported in part by FAPESB under grant 7320/2007 and by CAPES under grant PROCAD - AST-BAHIA 0271-05-5.

strengths and minimize the effects of their weakness. The paper is structured as follows: Section 2 presents an overview of both discussed approaches, Section 3 shows the experimentation conducted and discusses the results, and Section 4 enumerates our conclusions.

2. Background

This section contains a brief overview of the Constant Bandwidth Server, and discusses the two adaptation approaches that are the focus of the experimentation, Feedback Scheduling and Optimization-Based Adaptation.

2.1. Constant Bandwidth Server

Among the reservation-based server approaches, the Constant Bandwidth Server (CBS) [2] has received special attention since it is capable of providing temporal isolation, which is specifically desirable for applications with high variability in their execution times. According to CBS rules, the fraction of processor time allotted for each server is never surpassed even if its tasks overrun. The CBS approach is also simple to implement since it is based on the well known EDF policy to schedule the servers [12].

More formally: Consider a system of n CBS servers $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$. Each server S_i is defined by the tuple (Q_i, T_i), where Q_i represents its maximum budget and T_i represents its period. Each server S_i's utilization is at most $u_i = Q_i/T_i$ and serves a specific set of tasks. In other words, a system constructed in this manner allocates, for each set of tasks attributed to server S_i, a constant bandwidth determined by u_i, providing temporal isolation between servers even in overload conditions. The budget of each server S_i is depleted as its jobs are executed. To guarantee that the server obeys its utilization limit, the deadline of S_i is postponed by T_i every time its budget is depleted. At the same time, full budget Q_i is restored. When a new job arrives in an idle CBS, budget restoration and deadline postponement depend on a condition calculated at run time. For more details refer to [2].

2.2. Optimization-Based Adaptation

Optimization-Based Adaptation is the use of optimization algorithms to perform the distribution of processor time between tasks. This approach is best applied in complex systems with various tasks (with various well-defined modes), where combinatorial explosion makes it impractical or even impossible to define processor allocation at design-time. The definition of modes and the attribution of values to each one of them is possible, though, and leads to an optimization problem that is small enough to be solved at run-time.

This approach has been discussed in various papers, with goals ranging from maximizing a designer-defined benefit function [7, 10, 15] to aiding the operating system in maximizing battery life on mobile systems [16].

The approach previously proposed by the authors in [7] models run-time adaptation for reservation-based systems (in particular those scheduled using CBS servers) using optimization formulations, where the objective is to maximize the overall system benefit. This is the approach used in the experimental analysis of the following sections.

A brief explanation of this approach follows. Since every server is scheduled according to EDF, 100% utilization can be attained when scheduling the servers in $\mathcal{S}$.

That is, the system is schedulable if and only if

$$\sum_{S_i \in \mathcal{S}} u_i \leq 1 \tag{1}$$

In the context of this work, we assume that the system/user can, at any time, request a reconfiguration of the parameters of the servers in $\mathcal{S}$. This kind of request is associated to changing the operation mode of the system. To accomplish this, the application indicates what processor time percentages U_i should be allocated to each server S_i and a benefit value A_i associated to such an allocation. The benefit value actually achieved depends on the bandwidth u_i the system is able to set. For example, by selecting a movie screen whose processing is being served by S_i, the user may be indicating that U_i should be given to S_i so that the benefit A_i can be attained. However, if the system is able to allocate $u_i < U_i$, the actual benefit will be less than A_i. Also, there is no extra benefit in setting $u_i > U_i$ and this should be taken into account by the reconfiguration procedure.

As can be noted, reconfiguring the server parameters (Q_i, T_i) requires solving an optimization problem, where equation (1) is one of the constraints. Several models with different applications are presented in [7], but for illustration, the Integer Programming (IP) formulation is shown.

It is assumed here that, for each server S_i, $\kappa(i)$ values for U_i exist. In other words, let $K_i = \{1, 2, \ldots, \kappa(i)\}$ and $u_{ik} = Q_{ik}/T_{ik}$, $k \in K_i$, define the kth configuration of S_i for each of which there is a benefit A_{ik}. More formally, the optimization problem that must be solved by the reconfiguration mechanism is given as follows:

$$PD: \; fd = \text{Max} \sum_{S_i \in \mathcal{S}} \sum_{k \in K_i} A_{ik} x_{ik} \tag{2a}$$

S. to :

$$\sum_{S_i \in \mathcal{S}} \sum_{k \in K_i} u_{ik} x_{ik} \leq 1 \tag{2b}$$

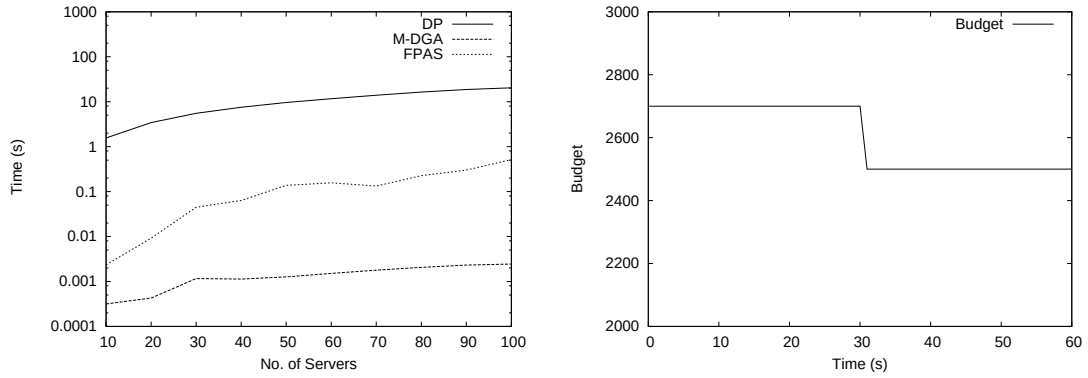
$$u_{ik} = \frac{Q_{ik}}{T_{ik}} \tag{2c}$$

$$\sum_{k \in K_i} x_{ik} = 1, \; S_i \in \mathcal{S} \tag{2d}$$

$$x_{ik} \in \{0, 1\}, \; S_i \in \mathcal{S}, k \in K_i \tag{2e}$$

The variable x_{ik}, defined by equation (2e), indicates which configuration $k \in K_i$ is chosen for the S_i server. Exactly one configuration must be selected, which is reflected in restriction (2d). Restriction (2b) guarantees the schedulability of $\mathcal{S}$ according to the EDF policy.

The cost of solving this optimization problem is shown in Figure 1(a). The modified density greedy algorithm (M-DGA) is a heuristic with performance bound $M\text{-}DGA(I) \geq \frac{OPT(I)}{2}$ for any given instance I of PD, where $OPT(I)$ is the optimal objective. The fully polynomial approximation scheme (FPAS) has a performance bound



(a) Execution Times of Algorithms for Different Problem Sizes.

(b) Optimization-Based Infrastructure Makes a Mode-Change.

Figure 1. Optimization-Based Adaptation.

$FPAS(I) \geq (1 - \epsilon)OPT(I)$ controlled by the parameter $\epsilon > 0$. The optimal dynamic programming (DP) algorithm is much slower than the other two. Figure 1(b) shows the instant nature of the Mode-Change, demonstrating that once the new scheduling parameters are known, they can be instantly applied. It is, of course, expected of the system designer to estimate the appropriate budget for each mode of the task, and how valuable they are for the system as a whole. After this is defined, actual mode changing is instantaneous.

2.3. Feedback Scheduling

In the context of CBS, there have been interesting developments in dynamically adjusting the server parameters by using feedback control theory [17, 1, 3, 14, 13]. The main idea of these approaches is to actuate on the server parameters, adjusting their bandwidth, so that a certain QoS metric is as close to the desired value as possible. For example, let the scheduling error of a job be the difference between its finishing time and its server deadline [1]. If the jobs of a task finish too late (too early), more (less) bandwidth should be allocated to its server. This approach is best applied when there is not enough knowledge about the temporal behavior of the system's tasks to make optimal processor allocations at design-time, and small, gradual variations in utilization are expected to occur dynamically.

To obtain the experimental results described over the next sections, the Feedback Scheduler described in Abeni *et al.* [3] was used. Since the dynamics of the system are simple, the authors of that work chose to use a PI controller to actuate on the parameters of the CBS.

Figure 2 shows the structure of the controller described in [3]. The scheduling error defined as the difference between Virtual Finishing Time (VFT) and the actual task period. The Virtual Finishing Time VFT_i of a job J_i is the time at which it would finish if executed on a dedicated processor of speed B_i. If the task finishes too early ($VFT_i < T_i$), the controller will give less bandwidth to it. If the task finishes too late ($VFT_i > T_i$), the controller will give it a bigger share of the processor.

The main difficulty in applying these approaches is determining a model for system dynamics so that a stabilizing control law can be designed. The choice of poles for

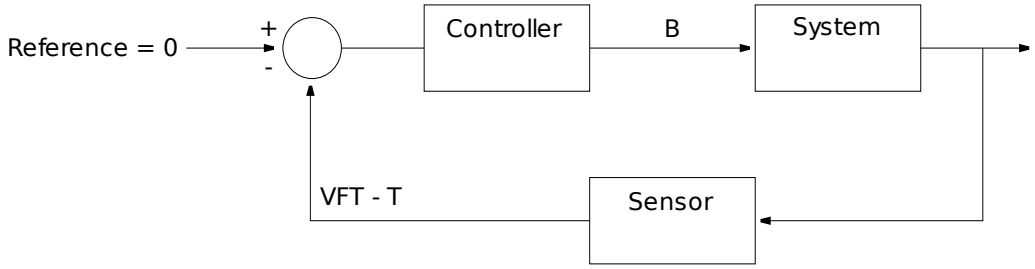
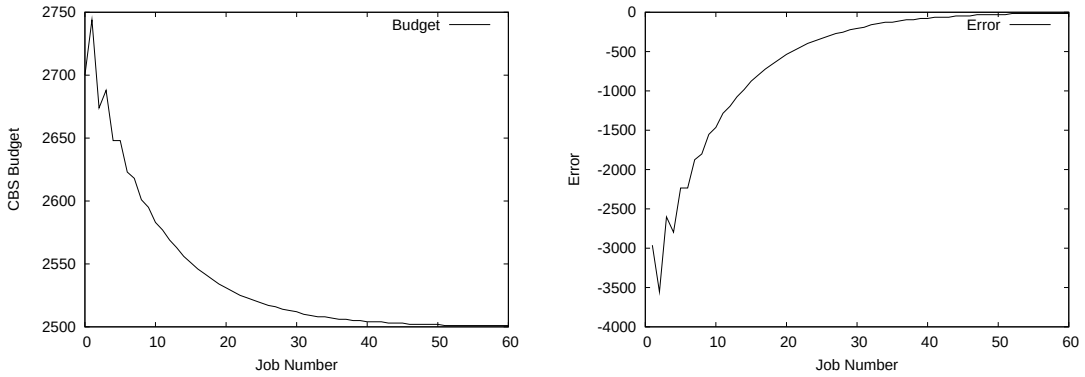


Figure 2. Feedback Control Applied to Reservation Scheduling

the controller may lead to an unstable controller or a restrictive controller that takes too long to stabilize at the reference level.



(a) Budget Control by Feedback Scheduling. (b) Error Measured by Feedback Scheduler.

Figure 3. Feedback Scheduling.

Figure 3(a) shows the Feedback Controller actuating on the budget of a CBS server to lower it from 2700 and stabilize it at the target value, 2500. Figure 3(b) shows the scheduling error gradually climbing to zero. The negative error means that the task was too early, and therefore should receive less processor time. Because of its conservative configuration, it took this controller 60 task periods to remove scheduling error.

2.4. Discussion

These two approaches have the common general goal of providing support for run-time adaptation of Reservation-Based schedulers. The first, Optimization-Based approach, does so by calculating which processor distribution is best for the system as a whole and then applies that configuration through an instantaneous mode-change. The second, Feedback Scheduling, corrects any scheduling errors such as lateness over time, trying to stabilize the server parameters at their correct values. Despite sharing the general goal, the application of these approaches is very different, since the Optimization-Based Adaptation requires significant knowledge of the application's temporal behavior by the system designer and a special API to allow run-time configurations, while the Feedback Scheduler approach does not. In the next section we discuss results of extensive simulation exploring some of the issues and benefits that come from the use of each approach and the opportunity of combining them in a more robust adaptive scheduling infrastructure.

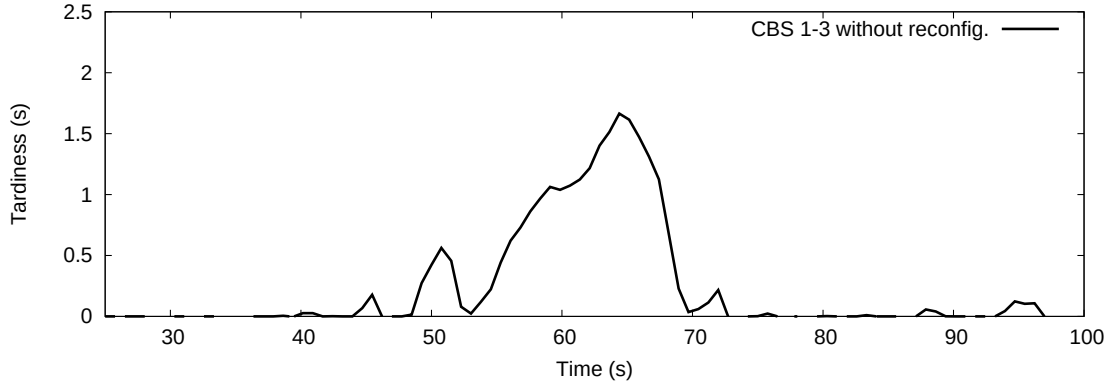
3. Experimental Analysis

This section presents simulation results from the Optimization-Based Adaptation Infrastructure presented in [7] and Feedback Control Scheduling presented in [3]. Although the values obtained from experiments are specific from these two approaches, our analysis is focused on general trends exhibited by the kind of adaptation carried out by them. Indeed, we use these two approaches as a means of analysis. All algorithms and simulations were implemented in C++ and executed on Linux on an Intel Core2Duo 2.20GHz CPU with 2GB of RAM. Since these benchmarks are synthetic, budget values are relative to a virtual period and do not necessarily represent any physical unit of time.

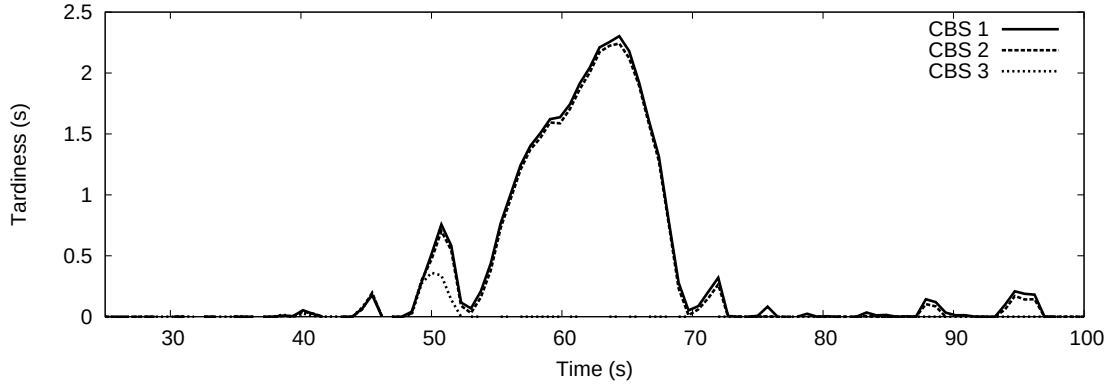
3.1. Optimization-Based Adaptation

For Optimization-Based Adaptation to function properly, the system designer must have a good estimation of each mode of the tasks and also define the value relationship between all the tasks. Results from [7], shown in Figure 4 for illustration, demonstrate success in diminishing the ill effects of overruns.

This system is comprised of three CBS servers, handling the exact same video decoding task. The utilization of the video task sometimes reaches 88%, meaning that deadlines will be missed. Consider that the importance of CBS 3 is, for some reason, higher than the importance of the other two servers.



(a) Tardiness Without Reconfiguration.



(b) Tardiness With Reconfiguration.

Figure 4. Optimization-Based Adaptation.

Figure 4(a) shows the system with no adaptation, with all servers missing the same deadlines with the same tardiness. Figure 4(b) shows the effect of shifting processor time to CBS 3. While CBSs 1 and 2 miss more deadlines and with higher tardiness, CBS 1 stops missing deadlines altogether (after a few periods, due to the backlog created by the overruns). This effect is a consequence of the benefit value defined for each task. When a reconfiguration was requested, a larger amount of processor time was dedicated to CBS 3, meaning that some had to be taken away from CBSs 1 and 2.

The optimization algorithm in this case took 19.5us to complete, about 0.2% of the time to decode a single frame of the video used in the simulation. A larger number of servers would mean a certain increase in this execution time, as seen in Figure 1(a). Also, it is not reasonable to believe that the system designer knows the exact parameters for all tasks at all times, therefore the budget value applied by the mode-change might not be optimal, that is, it may represent only an approximate knowledge of the system/environment. The fine-tuning of these parameters is better handled by a Feedback Scheduling scheme.

3.2. Feedback Control

Feedback Control, while requiring much less prior knowledge about the tasks' behavior, means that stabilization of budgets at the correct values is a longer process, affected by various factors, such as how large the mode-change is and how aggressive the configuration of the controller is. The following subsections present simulation results demonstrating how these factors affect the performance of the Feedback Scheduler described in [3].

3.2.1. Convergence Delays

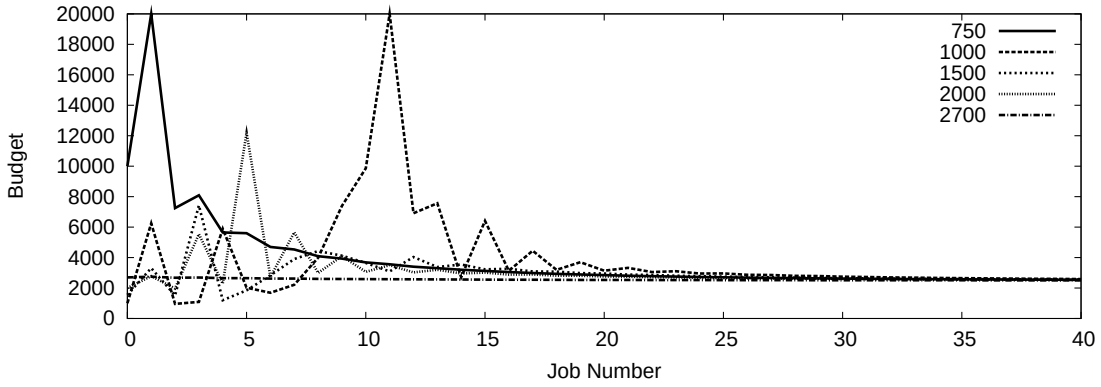
Generally, the larger the scheduling error, the longer the Feedback Scheduler will take to neutralize it, specially when configured conservatively. This is shown in Figure 5, where the Feedback Scheduler tries to stabilize the budget at 2500 from 5 different starting points.

The case with the starting budget closest to the reference, also shown in Figure 3, starts from an initial budget (B_1) of 2700 and takes 60 periods to stabilize at the target value. Table 1 shows the number of periods it takes to converge at the target value for the different values of B_1. The case where the initial budget $B_1 = 1500$, for example, oscillates around the equilibrium point with positive error for 12 periods, then oscillates with negative error, then finally converges on the target value.

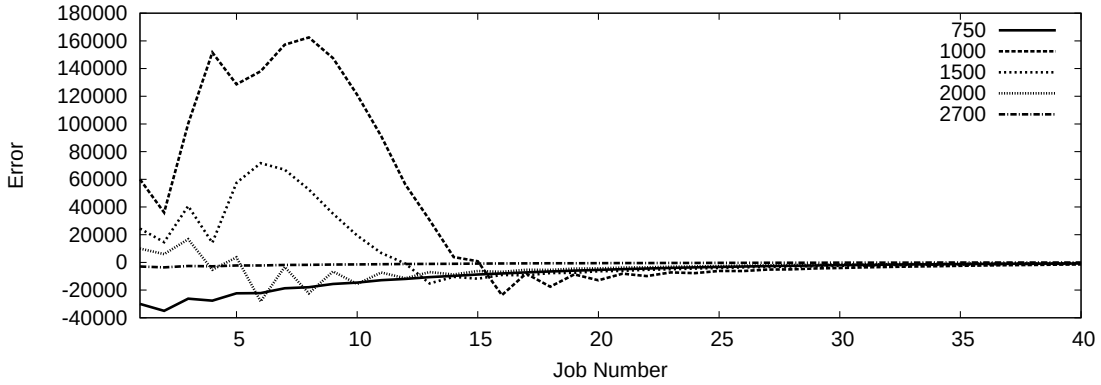
Table 1. Periods to Stabilization for Different Starting Values.

Starting Budget	750	1000	1500	2000	2700
Periods to Stabilization	84	91	86	81	60

Another product of the distance to the reference value is the increased oscillation, which can be very problematic for Real-Time Scheduling. In the cases where B_1 was 750 and 1000, control saturation occurred, leading to a reservation of 100% of processor time to be given to a task at some points. On the other hand, the fact that the case with $B_1 = 750$ converged faster $B_1 = 1000$ may be attributed to the early saturation of control, which



(a) Budget Control by Feedback Scheduling.



(b) Error Measured by Feedback Scheduler.

Figure 5. Feedback Scheduling Under Different Disturbance Values.

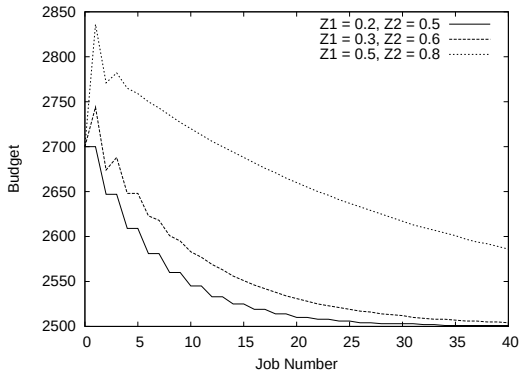
lead to the quicker stabilization of the budget value. If the pattern of mode-changing is known by the system designer beforehand, the problem of handling large changes in scheduling parameters is better handled by a scheme that is able to perform large mode-changes.

3.2.2. Pole Definition

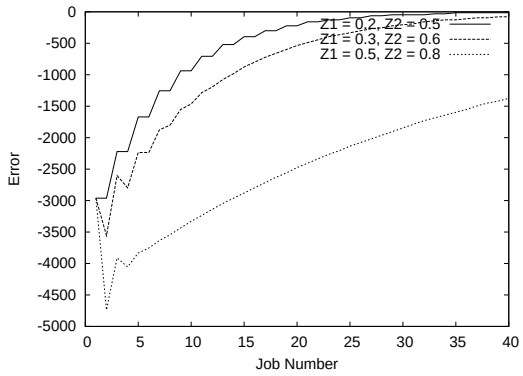
The definition of the poles for discrete time Feedback Controllers affects the gain values and therefore the speed of the controller. The control design presented in [3] places the poles of the closed-loop system closer to zero, thereby synthesizing a controller that reacts quickly and potentially "overreacts" to the measured error.

Figures 6(a) and 6(b) show the controller with different pole configurations controlling the same system. It is clear that the smaller the values are, the faster the controller is. When the poles Z_1 and Z_2 are set to 0.2 and 0.5 respectively, the number of periods until the system is stabilized with no error is less than 25% than that when $Z_1 = 0.5$ and $Z_2 = 0.8$.

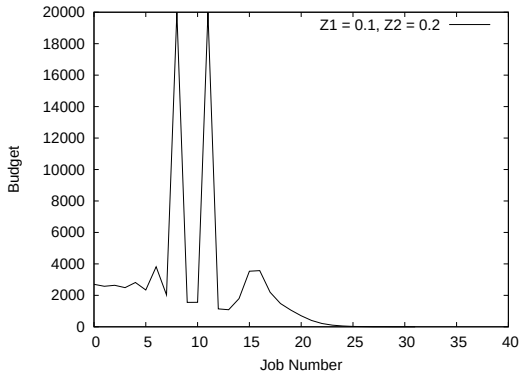
To illustrate a controller that is too aggressive, the simulation presented in Figures 6(c) and 6(d) show that there is a limit to how fast the controller can be, and the placement



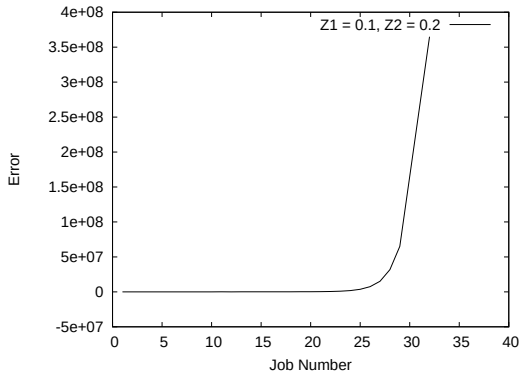
(a) Budget Control With Different Pole Settings.



(b) Error With Different Pole Settings.



(c) Failed Budget Control With Aggressive Poles.



(d) Error With Aggressive Poles.

Figure 6. Successful and Failed Feedback Scheduling Under Different Pole Settings.

of the poles does require some previous knowledge of the system. Due to the aggressive nature of the controller, the system oscillated to the point of setting the task budget value to zero, causing erratic behavior.

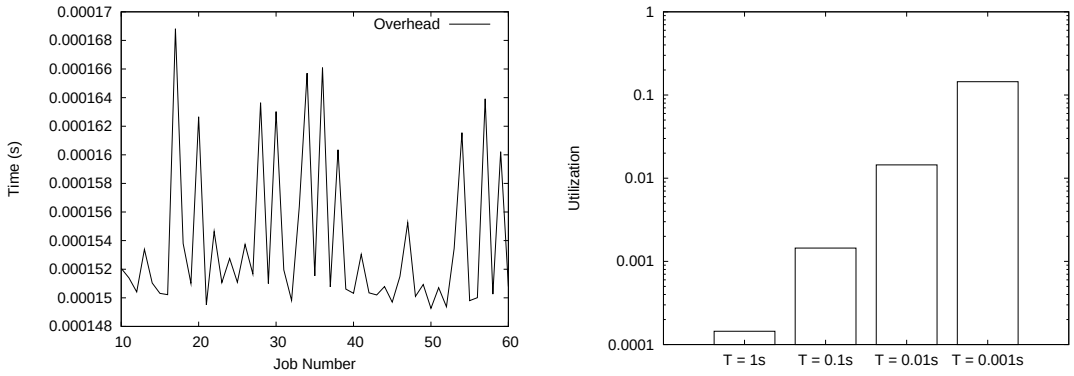
3.2.3. Overhead

Figure 7(a) shows the amount of time spent performing calculations for the Feedback Controller for each task period. Since only a few simple operations are needed each for period, the overhead of the Feedback Scheduler is very small, with average overhead equal to a tenth of a millisecond per period.

Since the calculations are performed once per period, the processor share that must be dedicated to the Feedback Controller is relative to the frequency of the task. Figure 7(b) shows that it would take a task with $T = 1ms$ to make the Feedback Controller use more than 10% of the processor.

3.3. Composition

While the aim of the optimization-based adaptation infrastructure shown in Section 2.2 is to provide a coarse adjustment of server parameters, the Feedback Controller does a better job at automatically correcting smaller scheduling errors. The integration of the



(a) Time Spent per Job on Feedback Scheduling. (b) Relative Overhead for Various Period Values.

Figure 7. Feedback Scheduling Overhead.

two approaches leads to a robust scheduling infrastructure that provides good support for applications with highly dynamic computation times and periods.

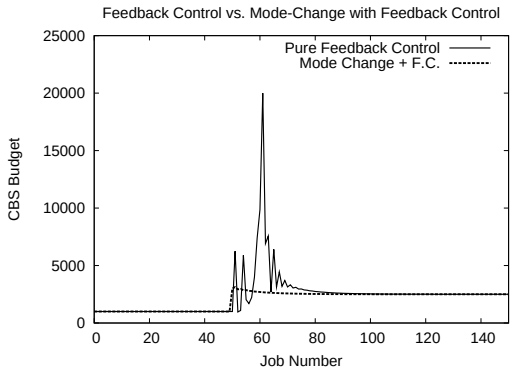


Figure 8. Composition of Mode-change and Feedback Control.

Figure 8 shows the integration of the two approaches [7]. Jobs 1 through 50 are scheduled without error with a CBS budget of 1000. Job 51 requires a budget of 2500, a step up in computation time. The feedback controller, when used by itself, oscillates and reaches the correct budget at job 142.

In the composition of a direct Mode Change and Feedback Control, the system designer overestimates the WCET and sets the budget for the new mode at 3000. The feedback controller then corrects this with much less oscillation and reaches the correct value of 2500 at job 120.

As can be seen from the figure, an adequate response to a drastic variation on utilization is achieved by the approach proposed in [7], while feedback control may be used to fine tune the scheduler to account for small variations in task utilization. The best performance is obtained by using both approaches in conjunction. While the optimization approach carries a high processing price for large server numbers, it is only executed once per large change in task utilization, while the feedback controller, which is executed once per task period, has very light processing needs.

4. Conclusion

This paper discussed some of the practical issues of bandwidth reservation adaptive schemes, namely Optimization-Based Adaptation and Feedback Scheduling. A specific approach to Feedback Scheduling [3] was implemented so that, through simulation, some of the benefits and issues of its use in Real-Time Systems could be explored. Properties such as the time until stability is achieved, the placement of the controller poles, and the overhead caused by its use were explored.

Furthermore, it was shown that the integration of Optimization-Based Adaptation systems and Feedback Scheduling leads to a good adaptive scheduling infrastructure, capable of handling very dynamic task periods and computation times. This was done through the integration of the infrastructure described by the authors in [7] and [6] and the Feedback Scheduling approach chosen for implementation.

References

[1] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proc. of the 6th IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA 99)*, pages 70–77. IEEE Computer Society, 1999.

[2] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[3] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. of the 23rd IEEE Real-Time Systems Symposium (RTSS 02)*, pages 71–80. IEEE Computer Society, 2002.

[4] G. Beccari, S. Caselli, and F. Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real-Time Systems*, 30(3):187–215, 2005.

[5] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.

[6] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration of constant bandwidth servers. In *Proc. of the 10th Brazilian Workshop on Real-Time and Embedded Systems (WTR 08)*, pages 37–44. Brazilian Computer Society, 2008.

[7] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)*, 2009.

[8] J. Jehuda and A. Israeli. Automated meta-control for adaptable real-time software. *Real-Time Systems*, 14(2):107–134, 1998.

[9] T.-W. Kuo and A. K. Mok. Incremental reconfiguration and load adjustment in adaptive real-time systems. *IEEE Transactions on Computers*, 46(12):1313–1324, 1997.

[10] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On quality of service optimization with discrete QoS options. In *Proceedings of the IEEE Real-time Technology and Applications Symposium*, pages 276–286, 1999.

[11] G. Lima, E. Camponogara, and A. C. Sokolonski. "Dynamic Reconfiguration for Adaptive Multiversion Real-Time Systems". In *Proc. of The 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS 08)*, pages 115–124, Prague, Czech Republic, 2008.

[12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogram in a hard real-time environment. *Journal of ACM*, 20(1):40–61, 1973.

[13] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, pages 13–13, 2000.

[14] L. Palopoli, T. Cucinotta, and A. Bicchi. Quality of service control in soft real-time applications. In *Proc. of the 42nd IEEE Conf. on Decision and Control*, pages 664–669, 2003.

[15] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical solutions for QoS-based resource allocation problems. In *IEEE Real-Time Systems Symposium,*, pages 296–306, 2000.

[16] C. Rusu, R. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *J. Embedded Comput.*, 1(2):271–283, 2005.

[17] P. Zhou, J. Xie, and X. Deng. Optimal feedback scheduling of model predictive controllers. *Journal of Control Theory and Applications*, 4(2):175–180, 2006.