Towards Adaptive Hierarchical Scheduling of Overloaded Real-time Systems

Nima Moghaddami Khalilzad, Thomas Nolte and Moris Behnam
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
nmi09001@student.mdh.se

***Abstract*—In a hierarchical scheduling framework, a resource can be shared among modules with different criticality levels. In our recently introduced adaptive hierarchical scheduling framework, modules receive a dynamic portion of the CPU during run-time. While providing temporal isolation is one of the main advantages of hierarchical scheduling, in an adaptive framework, for example when the CPU is overloaded, the higher priority modules can violate timing guarantees of the lower priority modules. However, the priorities of modules are assigned based on parameters other than the module criticality levels. For example the priority is often assigned according to periods and deadlines of tasks to increase the CPU utilization assuming static systems, i.e. modules parameters do not change during runtime. In an overload situation the high criticality modules should be superior to the low criticality modules in receiving resources. In this paper, extending our adaptive framework, we propose two techniques for controlling the CPU distribution among modules in an overload situation. We are taking another step towards having a complete adaptive hierarchical scheduling framework by incorporating an overload controller into our framework.**

I. Introduction

Adaptability is increasingly becoming the ubiquitous feature of real-time systems. Due to the fact that execution time of real-time tasks often varies significantly, adaptive scheduling of tasks is of importance. For example a decoder task of an H264 stream can experience more than five times execution time variation depending on the video content [1]. In order for resources to be used efficiently, systems should adapt themselves to the current load situation. We have recently introduced an Adaptive Hierarchical Scheduling Framework (AHSF) in which subsystems can adapt their bandwidth request according to the current load of their internal tasks [2]. Hence, the CPU resource is flexibly shared among subsystems which is conducive to a better performance of our adaptive framework in comparison with the static frameworks especially when tasks experience a sudden change in their execution time.

While, a number of studies have been conducted on overload scheduling [3], [4], [5], scheduling of mixed criticality systems in overload situations is also investigated in [6]. In this paper we study some applicable techniques that can be applied in the context of our Hierarchical Scheduling Framework (HSF) [7] in which modules budgets are adapted during run-time using feedback control loops [2].

Using the HSF we can provide temporal isolation for real-time tasks and guarantee their timing requirements [8]. In the HSF a system is composed of a variety of modules denoted as subsystems. During run-time, each subsystem receives a portion of the CPU and using this portion it should schedule its own tasks. All subsystems are scheduled using a global level scheduler.

The remainder of this paper is organized as follows. Section II provides a brief information about our HSF. Section III describes our AHSF. In Section IV two methods are suggested for dealing with scheduling in an overload mode. In Section V we show one example for illustrating the presented methods. Related works are presented in Section VI. Finally, our conclusions are presented in Section VII.

II. The Hierarchical Scheduling Framework

In this paper we consider the use of feedback scheduling in a single CPU where each CPU is modeled as a system S. Each system consists of a set of subsystems $S_S \in S$. The system is scheduled using a two level HSF. During run-time, the global scheduler chooses one of the subsystems and allocates CPU to that subsystem. Then, the subsystem's local scheduler shares this allocated CPU among its tasks according to its scheduling algorithm. In this paper, we study systems that are using the fixed priority algorithm in both local and global levels.

A. Subsystem Model

Each subsystem S_S is represented by its timing interface parameters (T_S, P_S, B_S, ζ_S) where T_S, P_S, B_S and ζ_S are subsystem period, priority, budget and criticality respectively. Each subsystem S_S also consists of a set of tasks τ_S and a local scheduler. Assuming n subsystems in the system there exist n levels of priority and criticality starting from 0 to $n-1$.

III. The Adaptive Hierarchical Scheduling Framework

In our AHSF, each subsystem has one budget controller which is responsible for adapting the budget of the subsystem to its internal tasks demands. The budget controller finds a suitable budget value for its corresponding subsystem by periodically sampling the controlled variables. The subsystem does not receive the new budget unless it is approved by the overload controller. The overload control logic is only activated in overload situations. In normal mode, subsystems can acquire their necessary budget values. The architecture of our AHSF is illustrated in Figure 1.

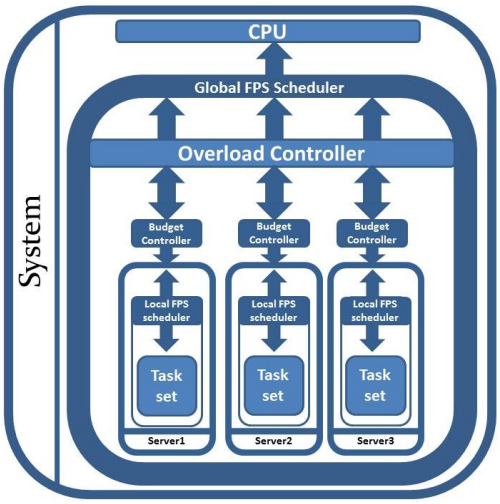

Fig. 1. Adaptive Hierarchical Scheduling Framework

The budget controller uses two PI feedback control loops for controlling the budget of subsystems. These loops are called "M-loop" and "U-loop" [2]. While the "M-loop" tries to minimize the number of deadline misses, the "U-loop" keeps the budget of subsystems to a minimum possible value.

IV. Overload Scheduling

In an overload situation, which we will call a *critical mode*, the controller cannot provide all subsystems with enough budgets. Therefore, we suggest a mechanism for distributing the CPU among subsystems based on their criticality value ζ_S. In dealing with the critical mode, the very first issue is detecting the time which the system mode is changing from normal to critical. Furthermore, after providing the most critical subsystem with enough budget, other subsystems should be able to use the remaining portion of the CPU resource such that they do not violate the schedulability condition of a higher criticality subsystem.

A. Mode Change

We suggest two mechanisms for detecting the mode change time. The goal of proposing these methods is to predict overload situations and to avoid critical deadline misses.

Both methods require conducting a schedulability analysis in global level. Since the global scheduler schedules subsystems in a similar way as scheduling simple real-time periodic tasks, it is possible to use the schedulability analysis methods used for scheduling periodic tasks. The subsystem can be modeled as a periodic task where the subsystem period is equivalent to the task period and the subsystem budget is equivalent to the task execution time. It is important to note that in the mode change methods we are interested in the global level schedulability, i.e, all subsystems should receive the assigned budget within their period. Therefore, a system can be schedulable in the global level and yet some tasks in the local level miss their deadlines.

1) Method one: According to offline analyses we can find a safe budget ceiling for all subsystems in which the whole system is schedulable and completely utilized. In doing so, we need to add an additional parameter to each subsystem in the subsystem interface which is the maximum budget value B_S^{Max}. Therefore, a system would be considered in its critical mode if any of its subsystems $S_S \in S$ is requesting a budget more than its maximum value ($B_S > B_S^{Max}$). It is important to note that changing the mode does not necessarily mean that there is not enough resources. We only force the system to do some additional checks by changing the mode.

In this approach, B_S^{Max} of the subsystems should be assigned to a safe value which could be a considerable safe boundary plus the time that all tasks inside that subsystem can finish their worst case execution time. This time duration can be calculated using the notion of real-time virtual processor model introduced by Mok et al. [9]. In calculating B_S^{Max} we should assume that all other higher criticality subsystems in the system $S_S \in S$ are using their B_S^{Max} and find a sufficient value for B_S^{Max} such that the response time of S_S is less than its period T_S.

2) Method two: Using an online schedulablity analysis in the global level, the system can detect the mode change time. Whenever the global scheduler fails to (analytically) schedule subsystems according to their new budget value, the system is considered to be in its critical mode.

B. Budget Distribution Policy in the Critical Mode

In the critical mode, the controller starts to share the budget among subsystems from the most critical subsystem to the least critical one. Therefore the most critical subsystem receives the entire budget that it requests. The amount of budget that the lower criticality subsystems receive is completely dependent on the new budgets of the higher criticality subsystems. If a lower criticality subsystem asks for less than the maximum possible budget it will get it, otherwise it will get the maximum possible value. It is an undeniable fact that in the critical mode, less critical subsystems might completely be shut down or receive a small amount of budget such that their tasks start missing their deadlines which is unavoidable. The important point to highlight here is that the criticality value of a subsystem should be assigned based on the criticality of its inner tasks. In the case that a subsystem is composed of mixed criticality tasks, one approach is to assign the average value of the tasks criticality to the subsystem criticality ζ_S and use a scheduler in the local level which takes the criticality level of tasks into account. In this approach subsystems should have a minimum budget value B_S^{Min} which indicates how much budget is necessary for only scheduling its corresponding high criticality tasks. Another approach is to simply assign the maximum criticality level of tasks to the subsystem criticality value. However, by using this approach the system discriminates between tasks that have the same criticality levels and belong to different subsystems.

C. Calculating the Remaining Budget

As it is mentioned, in the critical mode after assigning the budget for a higher criticality subsystem, there would be a limitation for the budget of a lower criticality subsystem. We present two approaches for mapping the consumed budget in a high criticality subsystem to the budget value of a low criticality subsystem. These approaches correspond to the methods presented for detecting the mode change.

1) Method one: If we are using an offline analysis for detecting a mode change, then we know B_S^{Max} for each subsystem, and we also know that if all subsystems use their B_S^{Max} the whole system is schedulable. In this method, after detecting the mode change, B_S^{Max} are assigned to their corresponding B_S.

For a system consisting of two subsystems S_i and S_j assuming $\zeta_i > \zeta_j$ and S_i requests a new budget that is α unit more than its maximum budget ($B_i = B_i^{Max} + \alpha$), the system enters to the critical mode and we initialize the budgets with the maximum budgets $B_S = B_S^{Max}$. In order to provide the high criticality subsystem with the requested budget we need to reduce the budget of the lower criticality subsystem. Hence, $B_j = B_j - \lceil \alpha \frac{T_j}{T_i} \rceil$. On the other hand, if afterwards S_i requests a budget value which is α unit less than its current budget B_i, we can transfer this extra budget to S_j using the following equation: $B_j = B_j + \lceil \alpha \frac{T_j}{T_i} \rceil$. These equations are used in implementation of the "TakeRequiredBudget(ζ_S, α)" and the "GiveExtraBudget(ζ_S, α)" functions presented in Algorithm 1 which shows pseudocode of method one. In this algorithm $NewBudget_i$ represents the new budget value that the budget controller suggests to the current subsystem S_i. When a subsystem requests a budget value which is less than its current budget we give the extra budget to the lower criticality subsystem using the "GiveExtraBudget($\zeta_i + 1$, α)" function. In the case that there is no other lower criticality subsystem in the system, this function reserves the extra budget in the lowest criticality subsystem such that the "TakeRequiredBudget(ζ_i, α)" function can use this spare budget.

The "TakeRequiredBudget(ζ_i, α)" function takes the required budget from some of the lower criticality subsystems (depending on amount of the requested budget) in such a way that S_i can receive the maximum available budget value. Indeed, when a subsystem asks for a budget value which is more than its current budget B_i, the controller takes this amount from the lowest criticality subsystem. If the lowest criticality subsystem cannot afford the whole required budget, the "TakeRequiredBudget" function gets the entire lower criticality subsystem budget and takes the remaining requested budget from the subsystem which belongs to the one level higher criticality (if its criticality is lower than criticality of the requested subsystem). The main purpose of exchanging budgets among subsystems in this method is to keep track of the overall available budget.

2) Method two: In this approach, we should do a schedulability analysis after each new budget assignment. In contrast with the global schedulability test which is done for mode change detection, in conducting the schedulability analysis for the subsystem S_i the algorithm assumes that all lower criticality subsystems are shut down. In doing so, when a higher criticality subsystem requires more budget, the algorithm punishes the lowest criticality subsystem. When the system is not schedulable, we have to rollback the last budget assignment and assign a lower value to the budget of that subsystem. Algorithm 2 shows pseudocode of method two. In this pseudocode the "$Schedulable(S,S_i)$" function conducts a schedulability analysis according to the new budget values. Furthermore, the $FindNewBudget(S_i,B_i)$ function returns a new value for the budget of subsystem S_i based on the last failed value.

Algorithm 1 Method one

```
for ζ_i = 0 to ζ_i = n − 1 do
    α = |B_i − NewBudget_i|;
    if NewBudget_i < B_i then
        GiveExtraBudget(ζ_i + 1, α);
        B_i = NewBudget_i;
    end if
    if NewBudget_i > B_i then
        B_i = B_i + TakeRequiredBudget(n − 1, α);
    end if
end for
```

Algorithm 2 Method two

```
for ζ_i = 0 to ζ_i = n − 1 do
    B_i = NewBudget_i;
    while Schedulable(S,S_i) ≠ True do
        B_i = FindNewBudget(S_i,B_i);
    end while
end for
```

V. Example

Assume a system with the specifications presented in Table I. In order to illustrate the introduced approaches, we present a scenario and show how we can apply these two methods to schedule the example system in the critical mode.

Name	T_S	Initial B_S	B_S^{Max}	P_S	ζ_S
S_1	20	1	2	1	1
S_2	22	3	4	2	2
S_3	18	2	2	0 (highest)	3
S_4	19	8	8	3	0 (highest)

TABLE I
SUBSYSTEMS SPECIFICATIONS

Assume that S_1 has current budget $B_1 = 2$ and that it requires two additional units of budget, and also assume that this will cause a mode change from normal to critical mode. The other subsystems have their initial budget values and they want to keep their budgets unchanged. If we want to schedule the system without considering their criticality, the timing constraints of S_1 are guaranteed only if the remaining budget after using S_3 is sufficient for S_1. However, since $\zeta_1 > \zeta_3$, in interfering of S_1 by S_3 we want S_1 to use the CPU.

A. Method one

- Since $B_1 > B_1^{Max}$ the system mode will change to the critical mode and $B_S = B_S^{Max}$.
- TakeRequiredBudget(3, 2) will take two units of budget from S_3 and will give it to S_1. Hence, $B_1 = 4$ and $B_3 = 0$.
- Since S_2 asks for a budget less than its current budget, the "GiveExtraBudget(3, 1)" will change $B_3 = 1$ and $B_2 = 3$.
- $B_3 = 1$ and S_3 asks for 2. Since there is no other lower criticality subsystem the "TakeRequiredBudget" function returns 0 and B_3 remains unchanged.

B. Method two

- The global schedulability check will fail (assumption) and it will cause a mode change.
- The budget of S_1 will change $B_1 = 4$ and the schedulability analysis assuming $B_2 = B_3 = 0$ will be successfully done.
- The budget of S_2 will be set $B_2 = 3$ and a schedulability test assuming $B_3 = 0$ will be done. Since the system is schedulable the algorithm will move to the next step.
- The budget of S_3 will be set $B_3 = 2$ and a schedulability test will be done. Since the system is not schedulable it will assign $B_3 = 1$ and perform the schedulability test again. Since this time the system is schedulable the algorithm will move to the next step.

VI. Related Works

Related works of this paper are twofold: adaptive scheduling and overload scheduling. Reservation-based algorithms are similar to our HSF in a sense that we also reserve resources for the subsystems. Feedback scheduling applied to the reservation-based algorithms [10]. Adapting the bandwidth of servers in reservation based scheduling algorithms is formulated as an optimization problem, and integer programming and linear programming solutions are suggested in [11]. Stankovic et al. have studied feedback control techniques in distributed systems [12]. Lu et al. introduced a Proportional (P) controller which controls CPU utilization requests based on miss ratio and utilization feedback loops [13].

de Niz et al. presented a scheme for protecting temporal isolation of high criticality tasks in mixed criticality systems. In their scheme a low criticality task cannot interfere with a high criticality task [6]. In [14] each task, in addition to a criticality value, has a mandatory and an optional part. In overloaded situations a set of task parts are chosen that maximizes the overall value of the system. In [15] the authors by introducing an elastic task model, showed that how tasks can adapted themselves to the different quality of services. Their proposed approach suggests that in overloaded situations instead of rejecting a new task by reducing the utilization of other tasks, the system lets the new task to use the CPU.

VII. Summary and conclusions

In this paper, we propose one online and one offline method for controlling the budget adaptation in an overload situation. When it comes to implementation, different approaches can be used for implementing these methods. For instance, the schedulability test can be done using either the utilization based test, the response time analysis or an approximation approach such as the one presented in [16].

The next step in our work is to implement these techniques in our simulation environment and further investigate pros and cons of each method. Moreover, we will conduct a set of experiments to compare the response time of tasks in AHSF and HSF. Finally, another trend of our work is to study other types of controllers instead of the PI budget controller.

References

[1] M. Åsberg, T. Nolte, C. M. O. Perez, and S. Kato, "Execution time monitoring in linux," in *Proceedings of the Work-In-Progress (WIP) session of 14th IEEE International Conference on Emerging Techonologies and Factory Automation (ETFA'09)*, September 2009.

[2] N. M. Khalilzad, M. Behnam, T. Nolte, and M. Åsberg, "On adaptive hierarchical scheduling of real-time systems using a feedback controller," in *Proceedings of the 3rd Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'11)*, April 2011.

[3] G. Buttazzo, M. Spuri, and F. Sensini, "Value vs. deadline scheduling in overload conditions," in *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, December 1995, pp. 90–99.

[4] S. Biyabani, J. Stankovic, and K. Ramamritham, "The integration of deadline and criticalness in hard real-time scheduling," in *Proceedings of 7th the IEEE Real-Time Systems Symposium (RTSS '88)*, Dec. 1988, pp. 152 –160.

[5] G. de A Lima and A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems," *Computers, IEEE Transactions on*, pp. 1332 – 1346, October 2003.

[6] D. d. Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS '09)*, December 2009, pp. 291–300.

[7] T. Nolte, M. Behnam, M. Åsberg, R. J. Bril, and I. Shin, "Hierarchical scheduling of complex embedded real-time systems," in *Ecole d'Ete Temps-Reel (ETR'09)*, August 2009, pp. 129–142.

[8] G. Lipari and S. Baruah, "A hierarchical extension to the constant bandwidth server framework," in *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTAS '01)*, May 2001, pp. 26 –35.

[9] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proceedings of the 7th Real-Time Technology and Applications Symposium (RTAS '01)*, May 2001, pp. 75–84.

[10] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a reservation-based feedback scheduler," in *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS '02)*, December 2002, pp. 71–80.

[11] A. B. d. Oliveira, E. Camponogara, and G. Lima, "Dynamic reconfiguration in reservation-based scheduling: An optimization approach," in *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications (RTAS '09)*, 2009, pp. 173–182.

[12] J. A. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu, "Feedback control scheduling in distributed real-time systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS '01)*, December 2001, pp. 59 – 70.

[13] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Systems*, pp. 85–126, 2002.

[14] P. Mejía-Alvarez, R. Melhem, and D. Mossé, "An incremental approach to scheduling during overloads in real-time systems," in *Proceedings of the 21st IEEE Real-time Systems Symposium (RTSS'00)*, November 2000, pp. 283–293.

[15] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS '98)*, December 1998, pp. 286 –295.

[16] N. Fisher and S. Baruah, "A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines," *Journal of Embedded Computing*, vol. 2, pp. 291–299, December 2006.