Towards Adaptive Hierarchical Scheduling of Real-time Systems

Nima Moghaddami Khalilzad, Thomas Nolte, Moris Behnam and Mikael Åsberg
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
nima.m.khalilzad@mdh.se

***Abstract*—Hierarchical scheduling provides a modular framework for integrating, scheduling and guaranteeing timing constraints of compositional real-time systems. In such a scheduling framework, all modules should receive a sufficient portion of the shared CPU to be able to guarantee timing constraints of their internal parts. In dynamic systems i.e., systems where the execution time of tasks are subjected to sudden and drastic changes during run-time, assigning fixed CPU portions to the modules is conducive to either low CPU utilization or numerous task deadline misses. In this paper, in order to address this problem, we propose an adaptive CPU allocation method which dynamically assigns CPU portions to the modules during run-time based on their current CPU demand. Besides, the presented approach is evaluated using a series of different simulations. In addition, we present a method for scheduling modules in situations when the CPU resource is not sufficient for scheduling all modules. We introduce the notion of module (subsystem) criticality, and in an overload situation we distribute the CPU resource based on the criticality of modules.**

I. Introduction

Increasing growth of the complexity in embedded real-time systems makes it difficult to combine real-time guarantees with efficient use of system resources. In real-time systems that do not have fixed run-time behavior, open loop scheduling techniques often result in poor performance (either there will be numerous task deadline misses or poor CPU utilization). In a dynamic real-time system, tasks can be added or removed during run-time, and their execution time can vary in a significantly vast range. A video decoder task is a prime example of a task that shows such a dynamic execution time behavior. For example, a decoder task of an H264 stream task can exhibit more than 500% variation in its execution time [1]. In such dynamic systems, closed-loop scheduling techniques can be used for adapting the scheduling parameters to the current behavior of the dynamic system. The system resources can be efficiently utilized using a feedback loop which observes the resource demand of tasks and adapts scheduling parameters according to the current load situation. Although a variety of previous studies have been performed on feedback scheduling [2], [3], [4], in this paper we design feedback loops which adapts scheduling parameters of our Hierarchical Scheduling Framework (HSF) [5].

The HSF is a component based approach for scheduling complex real-time systems [6], [7]. It can be illustrated using a tree structure in which each node is responsible for scheduling its own children using CPU resources received from its parent node. The child node provides its corresponding parent with interface parameters, such as period and budget, representing the resource required to guarantee schedulability of the child node. If the parent provides its child with the required budget within the required period, the child node will guarantee its timing constraints. One approach for finding resource efficient interface parameters for a child node is to assume a fixed period for it and try to find a minimum possible value for the budget [8]. However, in a dynamic system, the resource demand of a child node can vary over time. Therefore, if we manipulate its budget according to its current resource requirements, we can utilize the resources more efficiently, and the node to a greater degree can fulfill temporal constraints of its children.

In this paper, we introduce a feedback controller which adapts the budget value of nodes during run-time. Given a particular node period, the controller keeps the budgets to a minimum while at the same time minimizing the number of potential deadline misses within a predetermined time-interval. While in static systems where modules have fixed budget values and timing guarantees of one module cannot be violated by other modules, in dynamic systems we can end up in an overload situation where modules can affect each other's timing guarantees. For example, when we use a priority based scheduler, if a higher priority module consumes the entire available resource, a lower priority module must shut down.

However, it is often the case that the higher priority module is not the most important one. In this paper, we also investigate overload situations and suggest one method for dealing with budget adaptation in an overload situation. We add the notion of module criticality (inspired by [9]) to the interface parameters, and in an overload situation the scheduler gives priority to higher criticality modules in receiving the resource.

The contributions of this paper are i) the design of a feedback control system for dynamic adaptation of the interface parameters in the HSF, ii) simulation studies investigating the performance of our solution, and iii) a method for handling CPU overload in our Adaptive Hierarchical Scheduling Framework (AHSF). The preliminary work was presented in [10], [11].

The remainder of this paper is organized as follows. Related work is presented in Section II. Section III provides an overview of our HSF. Section IV describes the structure of the budget controller. In Section V a method is suggested for dealing with scheduling in overload mode. In Section VI we present simulation results and show one example illustrating overload scheduling. Finally, our conclusions are presented in Section VII.

II. Related Work

We categorize related work in three groups: hierarchical scheduling, feedback scheduling and overload scheduling.

A. Hierarchical Scheduling

Since Deng and Liu [6] presented a two level hierarchical scheduling framework, there has been a growing attention for using hierarchical scheduling in complex real-time systems. Schedulability analysis for two-level frameworks is presented by Kuo and Li [12]. For EDF-based global schedulers analysis is presented by Lipari and Baruah [13], [14]. In addition, the virtual processor model is presented in [15], [8]. Then, based on this model schedulability analyses under fixed priority scheduling [16], [17] and EDF [18], [8] are studied. While all of the aforementioned works allocate static CPU portions to the subsystems, we introduce an adaptive HSF which dynamically allocates the CPU to the subsystems.

B. Feedback Scheduling

Addressing control tasks with unpredictable execution time, feedback scheduling techniques have been efficiently applied [19], [20]. Reservation based algorithms are similar to our HSF solution, as we also use reservation based scheduling. The difference between our work and reservation based scheduling is that we have local schedulers in each subsystem. Therefore, tasks and subsystems are not necessarily scheduled using a same policy. Hence, a hierarchical system has a different behavior comparing with a flat system. Feedback scheduling has been applied to reservation based algorithms in [21], [22]. Bandwidth of the servers are adapted in order to improve application level and system level Quality of Service (QoS) using a two level feedback controller [22]. Feedback scheduling have also been applied in the distributed systems domain [4]. Lu et al. controlled CPU utilization request using a deadline miss ratio feedback loop [3]. Thereafter, they presented a two feedback loop system which in addition to the miss ratio, uses a CPU utilization feedback loop [2]. In this paper, however, we bring feedback scheduling techniques to the context of our HSF.

C. Overload Scheduling

In [9], using the notion of task criticality, the authors suggest a scheme for protecting temporal isolation of high criticality tasks. In [23] tasks consist of mandatory and optional parts. When the CPU resource is not sufficient, a sub set of task parts are chosen such that the value of the system is maximized. Buttazzo et al. presented an elastic task model in which tasks can adapt themselves to the current QoS. In their approach, whenever the CPU resource is not sufficient, instead of rejecting new tasks, the utilizations of accepted tasks are reduced such that also the new task can use the CPU. In [24] a technique for dealing with an overload situation using (m, k)-firm guarantee is proposed. The approach is suggested for the real-time control tasks that can tolerate occasional deadline misses. In this paper we study overload scheduling in our adaptive hierarchical scheduling framework.

III. The Hierarchical Scheduling Framework

A hierarchical scheduling framework can be modelled as a system S consisting of number of modules (subsystem) $S_S \in S$. In this paper we investigate a two-level HSF in which a global scheduler schedules subsystems, and each subsystem has its own local scheduler which is responsible for scheduling its internal tasks. Figure 1 shows the architecture of our two-level HSF which uses the Fixed Priority Scheduling (FPS) algorithm in both local and global levels. In this figure, the budget controllers and the overload controller are embedded in the architecture of the HSF.

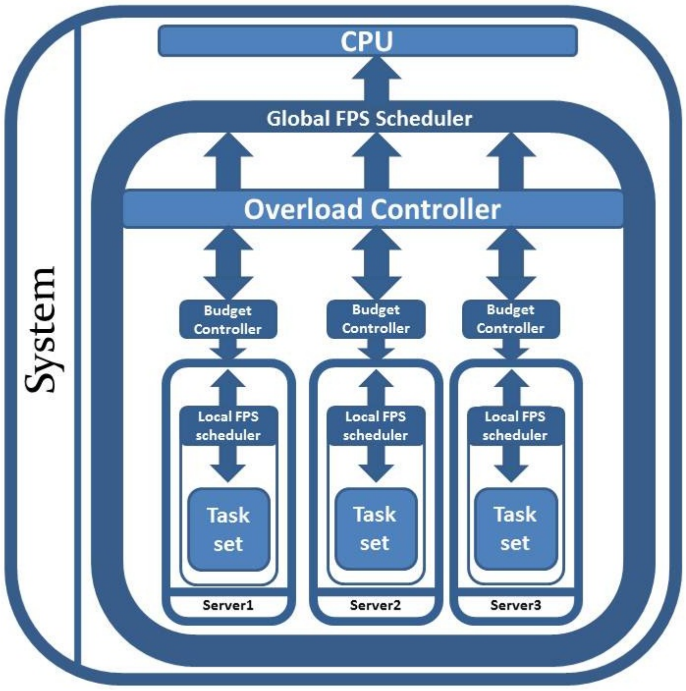

Fig. 1. Adaptive Hierarchical Scheduling Framework

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 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. The criticality of a subsystem ζ_S is used only in overload situations. Therefore, we ignore this parameter when systems are not overloaded. To keep the utilization required of subsystems as minimum as possible, we set the subsystem period to at most half of its shortest task period and then we find the minimum required budget for guaranteeing subsystem schedulability [25]. In each subsystem period T_S the subsystem receives a budget equal to B_S. The budget of subsystems should be set to a minimum, otherwise subsystems will experience idle time and resources will be wasted. Dewan et al. have proposed an algorithm for finding an exact minimum possible budget for subsystems that are using FPS in the local scheduler [26].

B. Task Model

We assume the periodic soft real-time task model $\tau_i(T_i, P_i, C_i, D_i)$, where T_i, P_i, C_i and D_i are task period, priority, worst-case execution time and relative deadline respectively. When a deadline miss happens in the system, the task continues executing until it finishes.

IV. Budget Controller

In this section we design a budget controller which manipulates the budget of subsystems according to the feedback that it receives from the scheduling plant. Therefore, the budget is our *manipulated variable* and we need to find proper *controlled variables* and sample them during run-time. In the design and analysis we are inspired by the work presented in [2]. However, we formulate the controlled system in the context of hierarchical scheduling.

The controlled variables are chosen according to two desirable features of a real-time system. First of all, in a real-time system the number of tasks that are missing their deadline should be either zero or as low as possible. In addition, available system resources should preferably be fully utilized. Therefore, we have two feedback loops in the system. The first loop observes the number of task deadline misses and manipulates the budget such that more tasks can meet their deadlines in the next sampling period. The second loop measures the amount of idle time in each subsystem, and manipulates the budget such that in the next sampling period this idle time becomes closer to zero. For simplification purpose we assume that these loops are independent from each other and we perform separate analyses for each loop.

A. Controlled Variables

The first feedback loop monitors task deadline misses. Therefore, we define the controlled variable of this loop $M_S(t)$ as the total number of missed deadline jobs of all tasks inside the subsystem $\tau_i \in S_S$, within one specified time window (tw_m) prior to the current time t. This control loop is called "M-loop" in the rest of the paper. The second feedback loop observes the idle time in the subsystems. In order to simplify analysis we do not directly use the idle time but instead define the second controlled variable based on the total available budget and amount of used budget. Hence,

$$U_S(t) = \frac{B_S(t)}{E_S(t)} \tag{1}$$

where $B_S(t)$ and $E_S(t)$ represent total budget of the subsystem and total measured time of CPU usage by all tasks of the subsystem S_S in the time window tw_u respectively. Similar to the M-loop, we call the second control loop "U-loop" in the rest of the paper.

B. Manipulated Variables

As it is mentioned, the budgets of the subsystems are manipulated variables of our controlled system. In each control period, the controller adds a budget change value $D_{B_S}(t)$ to the previous value of the subsystem budget.

$$B_S(t) = B_S(t-1) + D_{B_S}(t) \tag{2}$$

C. Integrating loops

In the presented architecture, each feedback loop in each control period has a budget change output $D_{B_S}(t)$. We choose a budget change value which has a greater absolute value than the others' absolute values. The reason for not using the smaller one is that when loops are in their saturation zone, their output is zero. For example when the subsystem idle time is equal to the set point but there are some deadline misses, the result of the U-loop is zero while the result of the M-loop is a positive number. Hence, a logical operation for integrating the results of the loops is to select the absolute maximum budget change value.

D. Model of Plant

In order to find the stability region of our controlled system we present an approximate analytical model of the plant. This model in combination with the model of the controller gives us model of the controlled system which is used in the stability analysis. In finding a plant model we are interested in the relation between the control output $D_{B_S}(t)$ and the control variables $U_S(t)$ and $M_S(t)$. From the definition of $U_S(t)$ we have

$$U_S(t) = \frac{B_S(t)}{E_S(t)}. \tag{3}$$

In order to continue analysis we assume that the subsystem tasks execute according to their worst case execution time $WCET_S = max(E_S(t))$. Hence

$$U_S(t) = \frac{B_S(t)}{WCET_S} \tag{4}$$

where $WCET_S$ is a total worst case execution time of all tasks in subsystem S_S. After transfering to the z domain, from the control input $D_B(z)$ to $U(z)$ the transfer function is:

$$U(z) = P_U(z)D_B(z) \tag{5}$$

and

$$P_U(z) = G_U/(z-1) \tag{6}$$

where

$$G_U = \frac{1}{WCET_S}. \tag{7}$$

We can define $M_{S_S}(t)$ based on U_{S_S} and derive a similar model for $M_{S_S}(t)$:

$$M_{S_S}(t) = M_{S_S}(t-1) + G_m(U_{S_S}(t) - U_{S_S}(t-1)) \tag{8}$$

where G_m is the deadline miss factor and can be found by plotting the $M_{S_S}(t)$ curve as a function of $U_{S_S}(t)$. For continuing the analysis we use G_M as the maximum value of

G_m. Similar to the U-loop we can derive the transfer function of the M-loop

$$P_M(z) = G_U G_M/(z-1). \tag{9}$$

E. Model of The Controller

We use a PI controller for both the M-loop and the U-loop to control the budget value of subsystems. The PI controller function is

$$D_{B_S}(t) = K_P Erorr_S(t) + K_I \sum_{tw} Errors(t) \tag{10}$$

where K_P, K_I, $Erorr_S(t)$ and tw are proportional gain, integral gain, error value of the subsystem S_S at time t and time window respectively. K_P, K_I are tunable parameters of the controller and should be tuned to get a desirable performance. $Erorr_S(t)$ is control loop error which is the difference between the controlled variable of each loop ($M_S(t)$ or $U_S(t)$) and the set point of that loop (M_{Set} or U_{Set}). After applying the z-transform we have

$$D_{B_S}(z) = K_P + \frac{K_I}{(z-1)}. \tag{11}$$

F. Closed-Loop System Model

If we consider $G = G_U G_M$ for the M-loop and $G = G_U$ for the U-loop, we can derive the following closed-loop system model for both loops:

$$H_S(z) = \frac{C(z)P(z)}{1+C(z)P(z)} = \frac{GK_P(z-1)+K_I G}{(z-1)^2+G(K_P(z-1)+K_I)} \tag{12}$$

G. Stability Analysis

From (12) the characteristic equation is:

$$(z-1)^2 + G(K_P(z-1)+K_I) = z^2 + \alpha_1 z + \alpha_2 \tag{13}$$

where $\alpha_1 = GK_P - 2$ and $\alpha_2 = 1 - GK_P + GK_I$. According to Jury's scheme [27, p. 82] the stability conditions are:

$$\alpha_2 < 1 \tag{14}$$

$$\alpha_2 > -1 + \alpha_1 \tag{15}$$

$$\alpha_2 > -1 - \alpha_1. \tag{16}$$

These conditions give us boundaries on the tunable variables of the system K_I and K_P.

H. Configurations

As shown in Figure 1, we have one budget controller corresponding to each subsystem. It implies that the controllers can be configured separately and they can have different periods, gain values and set points. Therefore, each controller should be configured according to the particular requirements of its inner tasks. As it is mentioned in Section IV-G, by applying the stability analysis we make the tuning process of the gain variables (K_P and K_I) easy. This section provides discussions about the tuning controller period and how to set the reference points in the feedback loops. However, we do not provide any mathematical analysis; we provide a discussion about the rationale behind selecting the controller period and the set points.

1) Tuning The Controller Period: In most of our simulations the controller period of the both feedback loops are selected to be two times larger than its corresponding subsystem period. The reason for choosing such a period is to let all subsystems work with their new budget so that in the next sampling we can measure the controlled variables that are resulting the new budget settings. The problem of choosing a larger period is that the M-loop cannot react to the deadline misses quickly, which is not desirable in some cases. A useful option that is possible in our AHSF is that we can set a relatively short control period for the critical subsystems and a relatively long period for the less critical ones. In summary, the controller period is an application specific tunable parameter which should be tuned according to the requirements of the subsystems.

2) Set Point of the M-loop: A desirable value for M is zero because in this case all subsystem tasks can meet their deadlines. However, since this value is on the threshold of deadline miss saturation, setting the reference point of the M-loop to zero results in a very sensitive controller which in some cases will be conducive to a large steady state error. Therefore, in subsystems that can tolerate some deadline misses by choosing the M-loop reference point to one or two, we can acquire a much better and stable performance.

3) Set Point of the U-loop: A desirable value for U is one because in this case all subsystems are using all CPU portions that they have received. Similar to the M-loop, since $U = 1$ is a threshold value, setting the reference point of the U-loop to one is conducive to a large steady state error. Therefore, a value between one and two is recommended for the set point.

V. Overload Controller

When the total requested budget of all subsystems in the system is more than the available CPU, the system is overloaded. In this situation, which we will call *critical mode*, if the controller provides all subsystems with their requested budget values, tasks that are in the lower priority subsystems will start to miss their deadlines. However, during critical mode the high criticality subsystems are superior to the low criticality subsystems. Therefore, in this section we introduce a method for distributing the CPU resource according to the subsystem criticality values ζ_S.

A. Mode Change

The budget controller, in each control period, suggests new budget values for the subsystems according to their current load situations. The overload controller can be consulted of the new budgets before they are assigned to the subsystems such that schedulability can be checked. When the budget controller wants to increase the budget value of any subsystems, the overload controller first performs a schedulability analysis given the new budgets. If the schedulability analysis fails, the system mode changes from the normal mode to the critical mode.

The schedulability analysis that is executed in the overload controller is performed on the 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 such periodic task systems. 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. We use the CPU utilization based schedulability analysis and, assuming that priorities have been assigned according to the rate monotonic algorithm in the global level, the schedulability test follows according to Liu and Layland [28]:

$$\sum_{i=0}^{n-1} \frac{B_i}{T_i} < n(2^{\frac{1}{n}} - 1) \tag{17}$$

where n is the number of subsystems in the system. Although the utilization based schedulability analysis is sufficient but not necessary which may lead to a case that the schedulability test fails and the overload controller punishes the lower priority subsystems even if the system may still be schedulable. We can avoid facing this potential problem by using an exact response time based schedulability analysis which on the other hand has higher computational complexity which is not preferred to be used during runtime.

B. Budget Distribution Policy in the Critical Mode

In the critical mode, the controller starts from the highest criticality subsystem and gives it a new budget. Thereafter, it moves to a lower criticality subsystem. It can only receive a budget value which is left after use of the highest criticality subsystem. This process continues until the lowest criticality subsystem receives the leftover CPU after all other subsystems are assigned to a new budget. In this approach a lower criticality subsystem tasks can start missing deadlines or even it can be completely shut down which is unavoidable. Indeed, the overload controller provides high criticality subsystems with enough CPU resource by punishing the lower criticality subsystems.

From Equation 17 and given that we have started to distribute the budget from the highest criticality subsystem to the lowest one, we can drive the following equation for the maximum available budget for subsystem S_i:

$$B_i^{Max} = T_i \underbrace{(n(2^{\frac{1}{n}} - 1) - \sum_{j \in S_h} \frac{B_j}{T_j})}_{U_{av}} \tag{18}$$

where S_h is the set consisting of all subsystems that have a higher criticality level than that of S_i, and U_{av} is the available CPU utilization for subsystem i. In this approach, if subsystem S_i asks for a budget value which is less than or equal to B_i^{Max}, it will be assigned that value, otherwise it will be assigned B_i^{Max}. Algorithm 1 shows pseudocode of the budget distribution policy in the critical mode. In this pseudocode $NewBudget_i$ represents a new budget that the budget controller suggests to subsystem S_i. Assuming that we have n subsystems in the system, there are n levels of criticality (from 0 to $n-1$) in the system. The algorithm loops through all subsystems from the lowest criticality level to the highest one, and using equation 18 assigns the new budgets to the subsystems. The computational complexity of this algorithm is of $O(n)$ which makes it suitable for dynamic systems.

Algorithm 1 Budget distribution policy

```
$U_{av} = n * (2^{\frac{1}{n}} - 1)$;
**for** $\zeta_i = 0$ to $\zeta_i = n-1$ **do**
    **if** $NewBudget_i \leq T_i * U_{av}$ **then**
        $B_i = NewBudget_i$;
    **else**
        $B_i = \lfloor T_i * U_{av} \rfloor$;
    **end if**
    $U_{av} = U_{av} - \frac{B_i}{T_i}$;
**end for**
```

VI. Simulation Results and Overload Example

For simulation purposes, we have used a synthesized scheduler according to the hierarchical scheduling model presented in [29]. We have synthesized the scheduler together with the budget control function and some functions for sampling the controlled variables. The controller function is embedded in the scheduler code such that the scheduler runs it (periodically) before each scheduling invocation.

A. Base Simulation

In order to show how the budget controller reacts to the execution time variation of tasks, a sample scenario is presented in this example. In this scenario the execution time of a task both increases and decreases during run-time.

The system consists of two subsystems. The specification of these subsystems is presented in Table I. Tasks that are in S_1 have fixed execution times and they use the entire available budget in each period. Table II shows the tasks that are in S_2. During run-time τ_1 experiences various execution time values. The execution time changes of τ_1 are presented in Table III. These execution time changes are done using a function which changes the value of execution time to a predefined value at a instant in time. All simulations presented in this paper are designed in a way such that the system has enough CPU resource to allocate to the subsystems. Therefore, we are only studying performance of our budget controller in these simulations.

Name	T_S	B_S	P_S
S_1	19	2	1
S_2	5	3	0

TABLE I
Subsystems specifications

In the base simulation the control period is 15, meaning that the controller samples the plant and manipulates the budget every 15 time unites. In the next simulation examples different values are assigned to the controller period.

Figure 2 shows result of executing the system for 600 time unites. The controlled variables M(t) and U(t) as well as the

Name	T_i	D_i	P_i	C_i
τ_1	10	6	1	3
τ_2	11	8	0	1

TABLE II
TASKS SPECIFICATIONS OF S_2

Time	0	50	200	400
C_1	3	2	3	0

TABLE III
EXECUTION TIME CHANGES OF τ_2

manipulated variable B(t) are sampled during run-time. The budget of S_2 is adapted during run-time as a result of the execution time changes of τ_1. The first budget change at time 30 occurs due to not sufficient initial budget of S_1. Afterwards, the budget has a similar change pattern as the execution time of τ_1 with some delay.

Whenever the execution time of τ_1 is increased, the system experiences some deadline misses which are detected by the M-loop, and this loop increases the budget of S_1. On the other hand, when the execution time of τ_1 is decreased, the U-loop observes the idle time in the subsystem and takes an action to reduce the budget value of S_1.

For comparing adaptive budget allocation with having a static budget, the system is executed using fixed budget values, and the result is illustrated in Table IV. The table shows the number of total deadline misses and the total idle time in S_1 using four different budget settings. This table shows a great improvement with respect to number of deadline misses when using our adaptive approach. Although the total idle time could be further reduced by using a lower set point for U-loop, since meeting deadlines is usually the most important objective in real-time systems, we have configured the controller in such a way that the system does not experience too many deadline misses.

Budget	4	3	2	adaptive
Deadline misses	0	12	33	4
Idle time	317	197	77	185

TABLE IV
IDLE TIME AND DEADLINE MISSES USING DIFFERENT BUDGETS

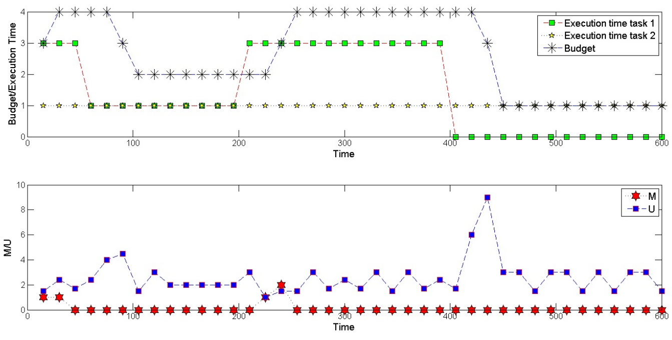

Fig. 2. Execution times, budget and controlled variables change over time

B. Different Configurations

In this part we set up the same system as the previous one but we change its configuration compared to the based simulation and we conduct some new simulation studies.

1) Control Period = 30: In this example, we are using the same system as the previous one with the difference in its controller period. The controller period is changed to 30 and the system is executed again. As it is shown in Figure 3 we can see a similar budget adaptation pattern as the base simulation but with some delay. In addition, the number of deadline misses is increased to six, compared to four in the base simulation.

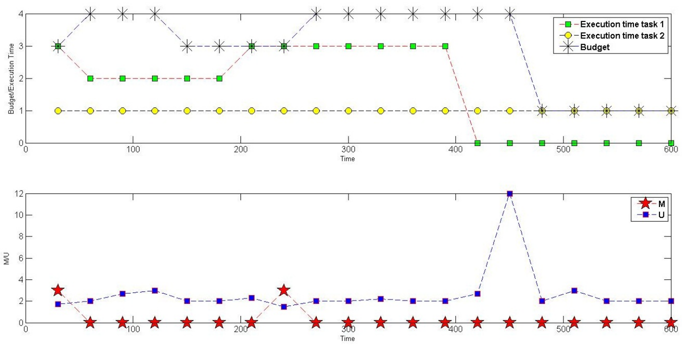

Fig. 3. Execution times, budget and controlled variables change over time (Control period = 30)

2) Control Period = 10: Here we set the controller period to 10 and execute the system. As shown in Figure 4, the number of deadline misses is decreased. Using this configuration the system experiences two total deadline misses, and the maximum number of deadline misses in one control period is one. The earlier the controller observes a deadline miss, the faster it can react. Note that when we are changing the controller period, we should also consider the control overhead on the scheduler.

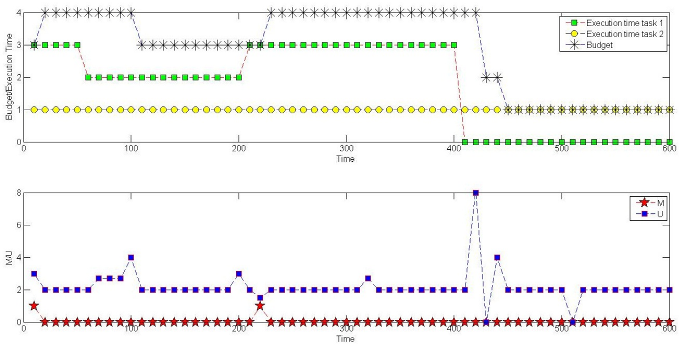

Fig. 4. Execution times, budget and controlled variables change over time (Control period = 10)

3) Different Set points: In this example the set points of the M-loop (M_{Set}) and U-loop (U_{Set}) are set to 0.5 and 1.2 respectively, and the corresponding result is shown in Figure 5. In the base simulation the set points were $M_{Set} = 0$ and $U_{Set} = 1.5$. Using this configuration the system experiences 10 total deadline misses, and the maximum number of deadline misses in one control period is two. On the other hand, the total idle time is 134 which is lower compared to the base simulation. This example shows that in the cases where a system can tolerate some deadline misses, we can configure M_{Set} to a value greater than zero and U_{Set} to a value close to one, then the system can achieve a high CPU utilization.

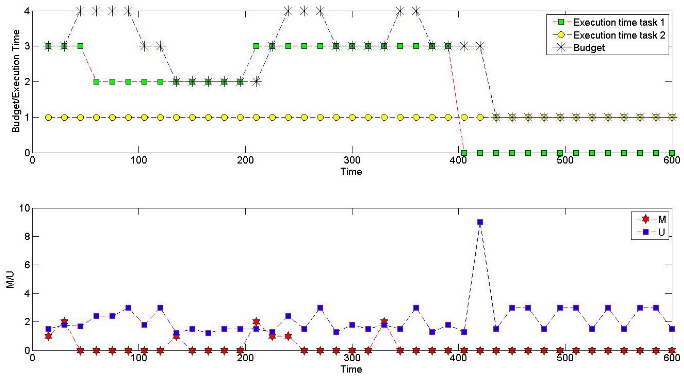

Fig. 5. Execution times, budget and controlled variables change over time ($M_{Set} = 0.5$ and $U_{Set} = 1.2$)

C. Four Subsystems

In this example we have four subsystems in a system. Table V shows the specification of these subsystems. Since the execution time of tasks inside subsystems are changing over time, the budgets of subsystems are adapted during run-time to improve overal system performance. Figure 6 indicates the relation between the controlled variables of the two feedback loops and the budget.

Name	T_S	Initial B_S	P_S	Number of tasks
S_1	23	1	1	4
S_2	20	2	0	2
S_3	20	2	2	2
S_4	31	1	3	1

TABLE V
SUBSYSTEMS SPECIFICATIONS

Task specifications of all subsystems are presented in Table VI. Budget adaptation in this example is inherent in two reasons. First of all, the initial budget values are not enough for some subsystems. In addition, the execution time of some tasks will change at time 200 and 300.

Subsystem	Name	T_i	D_i	P_i	C_i
S_1	τ_1	48	30	0	1
S_1	τ_2	48	40	1	1
S_2	τ_1	40	32	0	1
S_2	τ_2	40	30	1	1
S_3	τ_1	40	35	0	1
S_3	τ_2	42	40	1	1
S_3	τ_3	44	34	2	1
S_3	τ_4	48	19	3	0
S_4	τ_1	60	30	0	2

TABLE VI
TASKS SPECIFICATIONS OF ALL SUBSYSTEMS

D. Overload Control Example

Although the overload scheduling method is not implemented in our simulation environment, we present here an

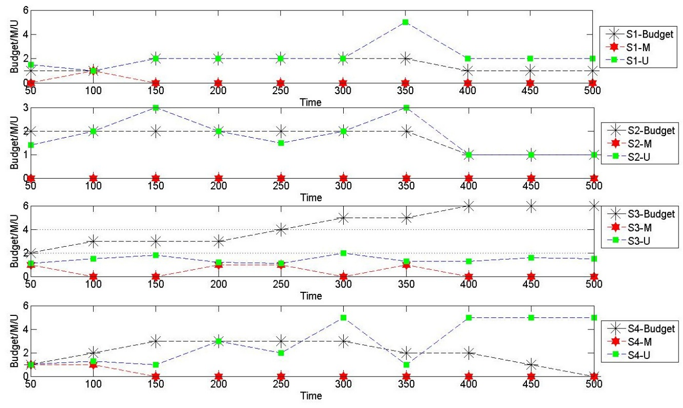

Fig. 6. Budget adaptation of four subsystems over time (S_i-M and S_i-U are controlled variables of the M-loop and U-loop respectively.)

Name	T_S	Initial B_S	P_S	ζ_S
S_1	15	2	1	1
S_2	17	3	2	2
S_3	14	2	0 (highest)	3
S_4	19	5	3	0 (highest)

TABLE VII
SUBSYSTEMS SPECIFICATIONS

example for illustrating the overload controller. Assume a system with the specifications as presented in Table VII.

Assume that during run-time the budget controller suggests 3 as a new budget of S_1. Assigning this new budget will put the system in an overload situation and therefore cause a mode change. We use Algorithm 1 to distribute the CPU resource among the subsystems as a result of this mode change, and:

- The available utilization is $U_{av} = 0.7568$.
- Since the utilization of the system is 0.7825 (assuming $B_1 = 3$) the overload controller will change the system mode.
- Iteration starts from the highest criticality subsystem which is S_4, and since $NewBudget_4 < T_4 * U_{av}$ $(5 < 14.3797)$, it can receive this new budget. The available utilization is updated: $U_{av} = 0.4937$.
- The next critical subsystem is S_1, and since $NewBudget_1 < T_1 * U_{av}$ $(3 < 7.4051)$, it can receive the new suggested budget, and $U_{av} = 0.2937$.
- The next critical subsystem is S_2, and since $NewBudget_2 < T_2 * U_{av}$ $(3 < 4.9924)$, it can receive the new suggested budget, and $U_{av} = 0.1172$.
- The last critical subsystem is S_3, and since $NewBudget_3 > T_3 * U_{av}$ $(2 > 1.6408)$, it cannot receive the suggested budget. Therefore, $B_4 = \lfloor T_3 * U_{av} \rfloor = \lfloor 1.6408 \rfloor = 1$.
- After the overload controller finishes its job, the budget values will be: $B_1 = 3$, $B_2 = 3$, $B_3 = 1$ and $B_4 = 5$.

The result of using Algorithm 1 is that budget has been taken from the lowest criticality subsystem S_3 and re-allocated to the higher criticality subsystem S_1. However, if we would not use the budget controller the global scheduler would prefer S_3 (the least critical subsystem) over all other subsystems.

VII. Summary and Conclusions

In this paper, we have studied feedback scheduling techniques in the context of hierarchical scheduling. Using a mathematical modeling approach, we have designed a PI controller which by sampling the number of deadlines and the amount of idle time in subsystems takes an action and manipulates the budgets. The performance of the budget controller is evaluated by simulation, the simulation results show that using the budget controller and re-allocating the CPU resource during run-time we can achieve a system with fewer deadline misses and higher CPU utilization.

In addition we have proposed an overload scheduling method for solving the problem of violating timing constraint of low priority subsystems, which may be side effect of manipulating budgets during run-time. In the presented method higher criticality subsystems are superior to the lower criticality subsystems in receiving CPU resources.

In compositional real-time systems where the execution time of tasks are not fixed during run-time, task execution times are not fully known before run-time, or tasks are added and removed during run-time, our adaptive hierarchical scheduling framework provides the system developers with an applicable solution for scheduling the real-time tasks.

The next step in our work is to implement the overload scheduling method in our simulation environment and study the performance of our adaptive scheduling approach in the overload situations. Moreover, we want to implement adaptive hierarchical scheduling on hardware, further evaluate our controllers and also measure the overhead of the controllers. Finally, another trend of our work is to investigate the presented approach in the context of multicore systems.

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] 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.

[3] C. Lu, J. Stankovic, G. Tao, and S. Son, "Design and evaluation of a feedback control edf scheduling algorithm," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, December 1999, pp. 56 –67.

[4] 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.

[5] 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.

[6] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," in *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, December 1997, pp. 308 –319.

[7] 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.

[8] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the 24th IEEE Real-Time Systems Symposium, (RTSS '03)*, 2003, pp. 2 – 13.

[9] 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.

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

[11] N. M. Khalilzad, T. Nolte, and M. Behnam, "Towards adaptive hierarchical scheduling of overloaded real-time systems," in *Proceedings of the 6th IEEE International Symposium on Industrial Embedded Systems (SIES' 11) , Work-In-Progress (WIP) session*, June 2011.

[12] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, December 1999.

[13] G. Lipari and S. K. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," in *Proceedings of the 6th IEEE Real Time Technology and Applications Symposium (RTAS '00)*, May 2000, pp. 166 –175.

[14] G. Lipari, J. Carpenter, and S. Baruah, "A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments," in *Proceedings of the 21st IEEE Real-time Systems Symposium (RTSS'00)*, November 2000, pp. 217–226.

[15] 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.

[16] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT '04)*, September 2004, pp. 95–103.

[17] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS03)*, July 2003, pp. 151–158.

[18] F. Zhang and A. Burns, "Analysis of hierarchical edf pre-emptive scheduling," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07)*, December 2007, pp. 423–434.

[19] A. Cervin and J. Eker, "Feedback scheduling of control tasks," in *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 5, December 2000, pp. 4871 –4876 vol.5.

[20] D. Henriksson, A. Cervin, J. Akesson, and K.-E. Arzen, "Feedback scheduling of model predictive controllers," in *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '02)*, September 2002, pp. 207 – 216.

[21] 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.

[22] T. Cucinotta and L. Palopoli, "Feedback scheduling for pipelines of tasks," in *Proceedings of the 10th international conference on Hybrid systems: computation and control (HSCC'07)*, April 2007, pp. 131–144.

[23] 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.

[24] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 6, pp. 549 –559, June 1999.

[25] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, pp. 30:1–30:39, April 2008.

[26] F. Dewan and N. Fisher, "Approximate bandwidth allocation for fixed-priority-scheduled periodic resources," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '10)*, April 2010.

[27] B. W. Karl Johan Astrom, *Computer-Controlled Systems: Theory and Design (3rd Edition)*. Prentice Hall, 1996.

[28] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, January 1973.

[29] M. Åsberg, P. Pettersson, and T. Nolte, "Modelling, verification and synthesis of two-tier hierarchical fixed-priority preemptive scheduling," in *Proceedings of the 23rd EUROMICRO Conference on Real-Time Systems (ECRTS'11)*, March 2011.