Adaptive Hierarchical Scheduling Framework: Configuration and Evaluation

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

Abstract—We have introduced an adaptive hierarchical scheduling framework as a solution for composing dynamic real-time systems, i.e., systems where the CPU demand of its tasks are subjected to unknown and potentially drastic changes during run-time. The framework consists of a controller which periodically adapts the system to the current load situation. In this paper, we unveil and explore the detailed behavior and performance of such an adaptive framework. Specifically, we investigate the controller configurations enabling efficient control parameters which maximizes performance, and we evaluate the adaptive framework against a traditional static one.

I. INTRODUCTION

The development process of a complex real-time system can be simplified by the usage of component based techniques. Such techniques naturally divide a complex problem into several less complex sub-problems following the divide-and-conquer principle. When applying component based techniques, then relatively simple components can be used to create a large and complex system. In this context, the timing constraints of each individual component as well as the composed system as a whole should be separately studied and guaranteed.

The Hierarchical Scheduling Framework (HSF) is a component based technique for scheduling complex real-time systems [1], [2]. Using such a framework, each component is allocated a portion of the CPU and, in turn, each component guarantees that with this portion all its internal tasks will be scheduled such that their corresponding timing constraints are respected. The system scheduler is responsible for providing the specified portion of CPU to all components, and the amount of CPU to be provided is often specified by the component period and budget (interface parameters) which indicate that each period the component should be provided CPU resources equal to the amount of the budget. These interface parameters can be calculated using different approaches depending on the scheduling algorithm used at local (within a component) and global (system) levels. For instance, one approach to calculate resource efficient interface parameters is to assume a fixed period and based on this period derive a minimum budget based on the Worst Case Execution Time (WCET) of tasks resident within the component [3]. However, in a dynamic system, since the resource demand of a component can vary significantly over time, the static budget allocation approach is inefficient. An example of dynamic tasks is found in the multimedia domain is among video decoding applications, where a video decoder task can experience large variation in execution time depending on the video currently being decoded. For example, video decoding applications, where a video decoder task can experience large variation in execution time depending on the video currently being decoded,

leading to oscillating execution times. Another example for dynamic systems is found in control systems applications, the computation of control tasks are dependent on values sensed from the environment. A third example of dynamic systems is found among the applications that scale the CPU frequency during run-time, which adds another source of variations in execution times of all tasks. In addition to changing resource requirements, some subsystems cannot be composed when their interface parameters are calculated based on tasks WCET due to the pessimism in their associated analysis, while, in practice, these subsystems can be composed to a single system if their corresponding CPU portions are adapted according to their run-time load demand. Hence there is a potential to increase applicability of hierarchical scheduling techniques to a broader range of applications, in particular applications found in the domain of multimedia.

As a solution to the above mentioned challenges we have introduced the Adaptive Hierarchical Scheduling Framework (AHSF) [4]. In this hierarchical framework we assume a fixed period for each subsystem, however, each subsystem is equipped with a budget controller which adapts the subsystem budget based on two feedback loops. The feedback loops are controlling the number of deadline misses and the amount of idle time in the subsystem.

In this paper we unveil and explore the detailed behavior and performance of the AHSF using the TrueTime simulation tool [5], and we present two contributions. First, we present a statistical approach for examining the system performance under different configurations and finding the best controller parameters. Secondly, we use the results obtained to examine the performance of the AHSF subjected to two dynamic load scenarios. To the best of our knowledge, this is the first study that rather than targeting the design of a controller for adaptive resource reservation schemes, targets the configuration of the controller in such schemes.

The remainder of this paper is organized as follows. Related work is presented in Section II. Section III describes the structure of AHSF. In Section IV we describe the simulation studies and compare the performance of HSF with AHSF. Finally, we conclude the paper in Section V.

II. RELATED WORK

Since Stankovic et al. introduced the idea of closed-loop real-time scheduling [6], there has been a growing interest in adopting feedback control techniques in the context of real-time scheduling. The deadline miss ratio is controlled using a PID controller in [7]. The idea is evolved in [8] which

uses two feedback loops: one controlling the deadline miss ratio and the other one controls the CPU utilization. Cervin et al. used a feedback-feedforward method for regulating the quality-of-control of the control tasks [9]. In addition they have investigated feedback scheduling of the control tasks [10] where the period of control tasks is adapted during run-time. Besides, a feedback scheduling technique for dealing with the variation of execution times in model predictive controllers is presented in [11].

There has been a growing attention to use resource reservation [12] and hierarchical scheduling [13], [14], [3], [15], [16], [17], [18] to schedule aperiodic, periodic, sporadic soft and hard real-time task models, as such techniques can provide a predictable behavior when integrating different task models, providing temporal isolation between different applications, allowing reusability and parallel development of subsystems in different applications, and such techniques also support open systems where applications can be added or removed during run-time. Besides, hierarchical scheduling is shown to be beneficial in scheduling of soft real-time systems in [19], [20], [21]. However, the basic assumption among all proposed analysis methods to compute resource allocation is that all tasks parameters are known in advance and they are fixed during run-time, i.e., an assumption which may not be true for many dynamic multimedia systems. Recently, many authors have targeted multimedia application and hence have attempted to enable the adaptability feature for resource reservation scheduling techniques by using feedback control techniques. Abeni et al. have investigated a reservation based feedback scheduling technique and a system model with analysis in [22]. They introduced an adaptive Constant Bandwidth Server (CBS) in which the server budgets are adjusted during run-time. This scheme is limited to existence of one task per each server. In order for it to work in the hierarchical framework where multiple tasks exist in each server, a new controlled variable should be defined. Adaptive CPU resource management is presented in [23] where the hard CBS scheduling algorithm is used and the server budgets are adapted during run-time. Moreover, application's Quality of Service (QoS) is adapted based on the available CPU resource. Although in theory this approach can support existing multiple tasks in the servers, the authors have evaluated their work only using one task in each server. In [24], in addition to the dynamic task execution times, the CPU temperature is considered as a dynamic hardware constraint which affect the total available CPU utilization. We have proposed the AHSF framework in [4] which uses a similar approach as the work presented in [8], but targeting hierarchical scheduling frameworks and also taking the problem of CPU overload into account by adding the criticality of modules (applications, subsystems, partitions) during online resource allocation.

In this paper, we focus on the AHSF, investigating its detailed behavior and performance by applying extensive simulation studies. The great impact of the framework configuration on the overall system performance manifested itself with the aid of a statistical tool which suggests that the configuration stage is of paramount importance in designing adaptive systems.

III. THE ADAPTIVE HIERARCHICAL SCHEDULING FRAMEWORK

We consider a two level Adaptive Hierarchical Scheduling Framework (AHSF) in which a system S consisting of N modules, here denoted as subsystems $S_s \in S$, is executed on a single processor. In the AHSF, a global scheduler schedules subsystems, and a local scheduler in each subsystem is responsible for scheduling its corresponding internal tasks. We have embedded two controllers in the AHSF to make it adaptive. The two controllers used in the AHSF are: i) the budget controller and ii) the overload controller. The controllers are explained later in this section. We use the EDF algorithm in both local and global levels.

A. Subsystem Model

Each subsystem S_s is represented by its timing interface parameters (T_s, B_s, D_s, ζ_s) where T_s , B_s , D_s and ζ_s are subsystem period, budget, relative deadline and criticality respectively. The relative deadline of a subsystem is assumed to be equal to its corresponding subsystem period ($D_s = T_s$). Each subsystem S_s consists of a set of n_s tasks τ_s and a local scheduler. The criticality of a subsystem ζ_s , which shows how critical a subsystem is in comparison to other subsystems, is used only in overload situations. Therefore, this parameter becomes very important when the CPU resource is overloaded. We assume that subsystems are sorted according to their criticality, in the order of decreasing criticality, and $\zeta_s = s$, i.e., S_1 has the highest criticality in the system while S_N has the lowest criticality. To keep the utilization required by subsystems to a minimum, we require the subsystem period to be at most half of the period of its shortest task period, and using this subsystem period 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 and any unused capacity of this budget will be consumed by an idle task in each subsystem¹. Let us define U^s as the system total utilization calculated based on the subsystem utilizations as follows:

$$U^s = \sum_{\forall S_s \in S} \frac{B_s}{T_s}. \quad (1)$$

Since we are using the EDF algorithm in the global level, the schedulability condition in our framework is $U^s < 1$.

B. Task Model

We assume the periodic soft real-time task model $\tau_{i,s}(T_{i,s}, C_{i,s}, D_{i,s})$, where $T_{i,s}$, $C_{i,s}$ and $D_{i,s}$ are period, WCET and relative deadline of task i in subsystem S_s respectively. The relative deadline of a task is assumed to be equal to its corresponding task period ($D_{i,s} = T_{i,s}$). When a deadline miss happens in the system, the task is aborted and a deadline miss is reported to the system. Let us define U^τ as the system

¹It might be more efficient to assign unused budget capacity to other subsystems, however, to keep the temporal isolation and to decrease the context switch between subsystem we have selected to idle the unused budget and still the budget can be decreased by the budget controller.

total utilization calculated based on the task set utilization as follows:

$$U^\tau = \sum_{(\forall \tau_{i,s} \in \tau_s) \wedge (\forall S_s \in S)} \frac{C_{i,s}}{T_{i,s}}. \quad (2)$$

Note that there is no clear relation between U^τ and schedulability and since the system is hierarchically scheduled, there is no guarantee for the schedulability of the system using existing analysis (for example the one presented in [3]) as such analysis are usually based on pessimistic assumptions.

C. The Budget Controller

The budget controller consists of two feedback loops. The first loop, which is called the "M-loop", monitors the number of task deadline miss instances in the corresponding subsystem. The total number of deadline misses, occurred between the last control period until current time t , is the *controlled variable* in this feedback loop. The controller suggests a new budget value for the subsystem based on the control variable at each control period.

The second loop, which is called the "U-loop", monitors the idle time in each subsystem (the budget that is not consumed by any of the subsystem's tasks). The *controlled variable* in this loop is defined as the unused budget in the last control period divided by the subsystem period which indicates the unused fraction of the assigned CPU for each subsystem. We use PI controllers in the feedback loops to calculate a budget change value. The PI controller function is:

$$D_{B_s}(t) = K_P Error_s(t) + K_I \sum_{tw} Error_s(t) \quad (3)$$

where $D_{B_s}(t)$, K_P , K_I , $Error_s(t)$ and tw are the budget change value of subsystem S_s at time t , proportional gain, integral gain, error value of the subsystem S_s at time t and time window respectively. $Error_s(t)$ is the control loop error which is the difference between the controlled variable of each loop (the number of deadline misses and the amount of idle time) and the set point of that loop.

D. The Overload Controller

At each time that a budget controller wants to increase the budget of a subsystem an overload test should be performed to make sure that all high criticality subsystems will receive enough budget. When the total requested budget of all subsystems in the system is more than the available CPU, the system is overloaded, we call this situation *overload mode*. The overload mode is detected using the following test:

$$U^s > 1. \quad (4)$$

During overload mode, if the controller would provide all subsystems with their requested budget values, and since the EDF scheduler is used, then some tasks from the same subsystem and different subsystems will start to miss their deadlines. However, during the overload mode the high criticality subsystems are considered superior to the low criticality subsystems. Therefore, the controller redistributes the CPU resource according to the subsystem criticality values ζ_s . It starts from the highest criticality subsystem S_1 and provides it with required budget. Thereafter, it moves to a lower criticality subsystem. The lower criticality subsystem can at most receive

a budget value which corresponds to the CPU resource that is left after allocation to the highest criticality subsystems. This process continues until the lowest criticality subsystem receives CPU resources, which happens after all other subsystems have been assigned a new budget. In other words, when the system is overloaded, the low criticality subsystems yield their budget to the high criticality subsystems. Using this greedy approach a lower criticality subsystem task can start to miss its deadlines, or in the worst case a subsystem can be completely shut down, which is unavoidable given the fact that the CPU resource is limited. Indeed, the overload controller provides high criticality subsystems with CPU resources by potentially punishing the lower criticality subsystems.

IV. SIMULATION STUDIES

We have used the TrueTime [5] simulation tool for evaluation purposes. The TrueTime kernel has been modified such that our AHSF has been implemented. We have utilized the TrueTime Constant Bandwidth Server (CBS) infrastructure to implement a two level hierarchical scheduler with EDF schedulers in both levels. In addition, some functions have been developed for implementing the controllers in the tool.

A. Simulation Setup

We have carried out two simulation studies to investigate the effect of different parameters on the performance of our AHSF. In these studies we also evaluate and compare the performance of the AHSF with a traditional non-adaptive HSF. The simulation studies have been performed by running a number of randomly generated systems in the simulator for a certain amount of time units, and we have measured the number of deadline misses that occur during this time.

Generating a random system is performed in two steps; in the first step utilizations are assigned to subsystems, where the input parameters are the number of subsystems to be created and their corresponding total tasks utilization. The total tasks utilization is divided among the subsystems using the UniFast algorithm [26]. In the second step, and for each subsystem, we use the assigned utilization to be divided among the set of tasks that belong to that subsystem, also using the UniFast algorithm [26]. The periods of the task set in each subsystem is selected as an integer number randomly from the range 1 to 20. Finally the execution time of each task is calculated given the task utilization and the task period. The subsystem period is set to a value equal to the half of the minimum task period among all tasks that belong to the subsystem, to guarantee that all tasks in the subsystem can be efficiently served [3]. Note that although assigning shorter periods to the subsystems can also serve the tasks, it imposes more context switch overhead to the system. In each evaluation stage we have different approaches for calculating the subsystem's initial budgets, which are explained later in the paper.

B. Study One: Effective Parameters

First we define the system total deadline miss ratio MR as a performance metric for comparing the effect of different configurations:

$$MR = \sum_{S_s \in S} w_s \frac{m_s}{f_s + m_s} \quad (5)$$

Fig. 1: Dynamic task ($T^{Chg} = 200$)

where, w_s , m_s and f_s are subsystem criticality weight, number of deadline misses and number of finished jobs of subsystem S_s in the previous control interval respectively. A control interval is the time duration from the last control action to the current time. In addition, the criticality weight for each subsystem is calculated using the following equation:

$$w_s = \frac{N - s + 1}{\sum_{j=1}^N j}. \quad (6)$$

We conduct two sets of simulations to investigate the effect of different system parameters on MR .

1) *Dynamic Task*: First we look into the systems that, in addition to some static tasks, contain dynamic tasks having execution time requirements that are changing over time. The changes in execution time of these dynamic tasks may push the system to the overload mode. To increase the effect of dynamic tasks on all other tasks we assume the existence of a dynamic task belonging to the highest criticality subsystem (S_1). The execution time of this task oscillates such that when it is high, the system is overloaded ($U^\tau = 1.2$), and when it is low the system is probably in the normal mode ($U^\tau = 0.5$). For mimicking the dynamic behavior, in both cases the task execution times are calculated using the normal random function with a mean value of $\frac{2}{3} \times C_i$. We define the change period of a dynamic task T^{Chg} as the duration that the dynamic task stays in one of its modes (either high or low). Figure 1 illustrates a dynamic task which changes its execution mode every 200 units ($T^{Chg} = 200$). This dynamic task model represents for example a multi mode task that changes its mode each T^{Chg} time unites.

Note that this dynamic task model that we consider in this section represents one class of the dynamic tasks. However, the approach that we use in the rest of the paper for investigating the effective parameters is not limited only to this class of dynamic tasks and can be applied to any arbitrary dynamic task class as well.

Table I summarizes the simulation settings that are used in this part. We have generated 50 random systems and each system is executed with all combinations of the settings. In total 10,000 different combinations are studied in this part and one task is selected randomly among the tasks that belong to S_1 to act as a dynamic task. The simulation is repeated for another 50 randomly generated systems to confirm that

Variable	Fixed value	min	max	Step size
$\#S_s \in S$	4	-	-	-
$\#\tau_{i,s} \in \tau_s$	3	-	-	-
T_i	-	1	20	random
U^τ	-	0.5	1.2	0.7
x	-	1	20	1
T^{Chg}	-	50	500	50

TABLE I: Simulation specifications

Fig. 2: Result of PCA

the first 50 samples are representative. Moreover, we have investigated the case of having two dynamic tasks in S_1 and also the scenario one (see Section IV-C2a) where there is a dynamic task in all subsystems. The agreement between the results of increasing the number of systems and the number of dynamic tasks is very good, however, these experiments are not presented in the paper.

In this study, we would like to investigate the effect of the budget controller period T_s^{Ctrl} and T^{Chg} on the total system deadline miss ratio MR in general and the specific subsystem deadline miss ratio mr_1 of S_1 which is computed as $mr_1 = m_1/(f_1 + m_1)$. In our AHSF, we select the period of the budget controllers to be proportional to their corresponding subsystem periods:

$$T_s^{Ctrl} = xT_s \quad (7)$$

where x is an integer value representing the control to subsystem period ratio. Note that we assume x is equal for all subsystems in the system. To analyze our simulation results we used Principle Component Analysis (PCA) which is a suitable statistical tool for studying the correlation among different variables [27], and in our case the variables are MR , mr_1 and ratios between T_s^{Ctrl} , T^{Chg} , T_s and minimum controller period T_{min}^{Ctrl} . Figure 2 shows the results of running PCA on our observed data. We study the angle between each pair of variables. The more closer to either 0° or 180° , the more correlated the variables. If the angle is between 0° and 90° the variables are positively correlated (both variables move in tandem), and if the angle is between 90° and 180° , the variables are negatively correlated (one variable decreases as the other one increases, and vice versa). Finally, if the angle is equal to 90° the variables are not correlated at all, so the closer the angle is to 90° the weaker the correlation.

Fig. 3: Total deadline miss based on x

The result from PCA shows that MR and x are more correlated than the MR and $\frac{T^{Chg}}{T_{min}^{Ctrl}}$. However, mr_1 is negatively correlated to $\frac{T^{Chg}}{T_1}$. These results suggest that the most important parameter (among all investigated parameters) affecting MR is the control to subsystem period ratio. Therefore, we remove all other parameters from our study and we focus on the effect of the control period T_s^{Ctrl} on MR .

In classic control theory, the fundamental assumption is that the controller is fast enough to control the system. It is known that the control frequency should be at least 10 times faster than the fastest dynamic element in the system. However, in the feedback scheduling literature, the controller frequency has not been emphasized enough. In such a system we want to control the system as slowly (with as low frequency) as possible while keeping performance of the system, in order to decrease the imposed overhead of the controller calculation to the CPU. Since the controller period plays an essential role in the system performance, we plot MR against x to see what ratio produces the minimum MR .

Figure 3 shows how the total deadline miss varies when the control to subsystem ratio changes. Mean values and standard deviations (bars) are presented in the figure (points are connected only to show the trend). The figure presents that a ratio between 2 and 4 produces the best results, however relatively large standard deviations suggest that there are some other parameters affecting the total deadline miss ratio that are not included in the first part of the study. Hence, we study the correlation among some other parameters in the next step. We can conclude from Figure 3 that the faster controllers do not necessarily produce the better results, because for instance when $x = 1$, the "U-Loop" controller does not let the subsystem to run with its new budget for enough time causing the control decisions to be based on immature sensor data. The figure indicates that there is no tangible difference among 2, 3 and 4. Therefore, for the sake of low overhead we choose $x = 4$ as the control to subsystem ratio based on this part of the study.

We repeat our simulations to explore the correlation between MR and a new set of parameters which are relative deadline

Fig. 4: Result of PCA # 2

Fig. 5: MR in different period levels against x

(which equals to task period in our case) of the dynamic task $D_{1,1}$, period of the dynamic subsystem (T_1) and utilization of the dynamic task ($U_{1,1}$). Considering the fact that the shorter the period of tasks and subsystem, the more likely to have a higher scheduling priority during run-time than the others, we investigate the relation between the periods and MR . In addition, we study the effect of standard deviation of subsystem periods (T_s^{std}) on MR . We run 1000 randomly generated systems with the same settings as in the previous part. The only difference is that the change period T^{Chg} is fixed and it is equal to 50. In addition we only explore x in the range of 1 to 10, based on the results that we got in the previous part. Figure 4 shows that among all investigated variables in the second PCA, the total deadline miss ratio is mostly correlated with the period (which is equal to the deadline) of the dynamic task. Therefore, we plot the mean values of the control to subsystem ratio for each relative period level P_i ($P_i = 1$ is the shortest period) separately in Figure 5. The figure indicates that the shorter the dynamic task period (compared with the other periods in its subsystem), the higher the total deadline miss ratio which is due to the greater impact of the dynamic task to the other tasks in its subsystem. The interesting point for us is that all priority levels are following a similar pattern which makes it possible to draw more confident conclusions. The figure also shows that when the dynamic task has the highest priority, the minimum total miss ratio is shifted to six, however, in all other cases it is still four.

Fig. 6: Total deadline miss in different x when the budgets are initialized zero

Fig. 7: Budget adaptations when the budgets are initiated to zero

2) *Zero Budget Subsystems*: To further study the effect of the controller period on the total deadline miss ratio, we design another type of simulations. In the initialization phase of our simulations we assign the budget value equal to zero for all of the subsystems and we let the controllers to adapt the budgets based on the load requirement of each subsystem. Note that all tasks are static in this simulation. We generate 50 random systems where $U^\tau = 0.5$, and for each generated system we explore x in the range of 1 to 15.

Figure 6 shows MR against x . We can see that when the ratio is between 2 and 4, the total miss ratio is the minimum which is consistent with the previous simulations. Figure 7 shows a sample system consisting of four subsystems in which the budgets are initiated to zero and they are adapted during run-time. This figure shows the adaptation speed in each subsystem subjected to full load as a step input.

3) *Discussion*: Our simulation results in this part show that the best control period depends on a variety of parameters, and we can not select a single control to subsystem ratio which provides the best performance for all systems. However, we conclude that the best control to subsystem ratio is most likely in the range of 3 to 6. Therefore, we choose the controller period 4 times larger than the subsystem period ($x = 4$) and we are aware that this ratio might not be the optimum ratio for some subsystems, nevertheless, it is close enough to the optimum ratio. We would like to highlight that $x = 4$ is derived based on the assumed dynamic task model present in Figure 1. Our contribution however, rather than the final value, is our

Fig. 8: Total deadline miss ratio of HSF and AHSF in different U^τ

approach in identifying the important parameters and finding the best configuration.

C. Study Two: Comparison between HSF and AHSF

In this section, we compare the performance of the AHSF and a traditional HSF (without adaptation), and for the AHSF framework and for each subsystem S_s we configure the controller period based on the conclusion drawn in the previous section, i.e., $T_s^{Ctrl} = 4 \times T_s$. For comparison reason, we will first show how our AHSF behave in terms of deadline miss ratio compared with a traditional HSF for the case of static systems (execution requirements do not change during run-time) that require more than 100% CPU resources, i.e., $U^s > 1$. Then we show the other case when the CPU resource requirements are changing dynamically, i.e., execution times of tasks are changing during run-time.

1) *Static Systems*: In the first part, we generate 100 random systems for each total tasks utilization U^τ in the range of 0.4 to 1.6 with 0.1 increment, and we run them in the simulator using both HSF and AHSF. In the HSF case, subsystem budgets are calculated using supply bound and demand bound functions [3]. Note that the schedulability of subsystems of each randomly generated system is not guaranteed, i.e., $U^s > 1$, however, we run the system and record the total deadline miss ratio. Figure 8 presents MR against U^τ . The unweighted total deadline miss ratio, i.e., MR where w_s is equal for all subsystems ($w_s = \frac{1}{4}$), is also shown in the figure (dotted lines). The difference between unweighted deadline miss ratio and weighted deadline miss ratio in the AHSF case is very low, however, in the HSF case the weighted is higher than the unweighted because the HSF does not take criticality levels into account. The figure shows that when $U^\tau < 1$, the AHSF can provide very low deadline miss ratio, and when the system is overloaded, still the AHSF has lower MR compared with the HSF. These result suggest that the AHSF can be used for finding and assigning interface parameters in hierarchical frameworks. For instance if a subsystem is added or removed during run-time, the AHSF can redistribute the budgets in the system in an efficient way.

2) *Dynamic Systems*: In this part, we show some simulation results on our target systems, i.e., systems that are

Fig. 9: Execution time changes of dynamic tasks in scenario one

composed of dynamic tasks and where their execution times can change during run-time. In addition, each subsystem has a unique criticality level meaning that the system's first goal is to reduce the deadline miss ratio for the most critical subsystem. We identify two different scenarios to be demonstrated in this section and for the case of AHSF, the budgets of subsystems are initialized according to the following equation:

$$B_s = \sum_{\forall \tau_{i,s} \in \tau_s} \left(\frac{C_{i,s}}{T_{i,s}} \right) \times P_s \quad (8)$$

while for the traditional HSF the budgets are calculated using the same approach as in the previous part (the approach proposed in [3]).

a) Scenario One: In this scenario each subsystem is composed of one dynamic task and two static tasks. The task execution times are changed according to Figure 9. The system is randomly generated with $U^\tau = 0.3$ and each dynamic task adds 0.3 to U^τ when it is in its high mode. Therefore, the system load varies such that at some points, e.g., time 850, the system is overloaded ($U^\tau = 1.2$).

Fig. 10: Budget adaptations in scenario one

Figure 10 shows the budget adaptation of the subsystems during run-time. An interesting point to highlight in Figure 10 is that S_4 is shut down when the higher criticality subsystems are in their high mode and for example from time 500 to 700 since the other subsystems are in their low mode, S_4 can get some resources. Furthermore, when S_1 and S_2 together are in their high mode, both S_3 and S_4 are suffering from low CPU resource availability.

Framework	MR mean	MR std	mr_1 mean	mr_1 std
HSF	0.1849	0.0660	0.0770	0.0499
AHSF	0.0710	0.0271	0.0273	0.0187

TABLE II: MR and mr_1 in scenario one for HSF and AHSF

Fig. 11: Execution time changes of dynamic tasks in scenario two

In addition, the scenario is tested on 100 randomly generated schedulable systems. Since task number one $\tau_{1,s}$ is always the dynamic task in all subsystems, it has a new random period $T_{1,s}$ and utilization $U_{1,s}$ in each generated system. The mean value and the standard deviation of MR and mr_1 are presented in Table II. As it is expected, the performance of AHSF is much better than HSF with respect to both total and S_1 deadline miss ratio.

b) Scenario Two: In the second scenario we remove the dynamic task from S_1 , besides, the execution times of the dynamic tasks in S_2 , S_3 and S_4 are changed according to Figure 11.

Figure 12 illustrates budget adaptation of a sample system under scenario two. Since there is no dynamic task in the highest criticality subsystem, the overload controller should guarantee enough resources for S_1 , i.e., the global scheduler supplies the subsystem with the required budget every subsystem period and the rest of the CPU resource should be distributed among the other subsystems based on their loads and criticality levels.

Fig. 12: Budget adaptations in scenario two

Table III presents the mean value and standard deviation of both MR and mr_1 for 100 randomly generated systems which are run under scenario two. Although, in the AHSF, S_1 is experiencing some deadline misses which is due to the optimistic budget initializations, the mean value of mr_1 is close to zero. Furthermore, the total miss ratio in the AHSF is much lower than the HSF.

Framework	MR mean	MR std	mr_1 mean	mr_1 std
HSF	0.1339	0.0559	0	0
AHSF	0.0532	0.0215	0.0016	0.004

TABLE III: MR and mr_1 in scenario two for HSF and AHSF

The overhead of the presented adaptation approach is negligible because i) the controller runs infrequently ii) it only executes simple instructions. A more detailed discussion on the overhead in addition to a multimedia case-study is presented in the technical report version of this paper [28].

V. CONCLUSIONS AND FUTURE WORK

In this paper, we first presented a general statistical approach for detecting parameters which influence the system performance when using our adaptive hierarchical scheduling framework. We show that for our assumed dynamic system the most important parameter is the controller period, and we found an efficient control to subsystem period ratio. Furthermore, we have evaluated our adaptive framework under different dynamic loads and we have compared its performance with a static framework. The simulation results show that the adaptive framework can effectively deal with dynamic loads in the system and it will reduce the deadline miss ratio especially for high criticality subsystems. The next step in our work is to implement the adaptive framework in Linux and to run some dynamic applications for further evaluation of the framework. We are also investigating other types of control techniques to use, such as event-based control, to replace our controller with a more effective one if it would show to be beneficial. Finally, we have an intention to extend our adaptive hierarchical scheduling framework to multi-core platforms.

REFERENCES

- [1] 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.
- [2] 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.
- [3] 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)*, December 2003, pp. 2 – 13.
- [4] N. M. Khalilzad, T. Nolte, M. Behnam, and M. Asberg, "Towards adaptive hierarchical scheduling of real-time systems," in *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '11)*, September 2011, pp. 1 – 8.
- [5] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen, "How does control timing affect performance? analysis and simulation of timing using jitterbug and truetype," *Control Systems, IEEE*, vol. 23, no. 3, pp. 16 – 30, June 2003.
- [6] J. Stankovic, C. Lu, S. Son, and G. Tao, "The case for feedback control real-time scheduling," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS '99)*, June 1999, pp. 11 – 20.
- [7] 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.
- [8] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Systems*, vol. 23, no. 1/2, pp. 85 – 126.
- [9] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzen, "Feedback feed-forward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1/2, pp. 25 – 53, July 2002.
- [10] A. Cervin and E. Johan, "Feedback scheduling of control tasks," in *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 5, December 2000, pp. 4871 – 4876.
- [11] 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.
- [12] C. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: operating system support for multimedia applications," in *Proceedings of the International Conference on Multimedia Computing and Systems*, May 1994, pp. 90 – 99.
- [13] 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.
- [14] 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.
- [15] 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, pp. 256 – 267.
- [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 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–.
- [18] 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.
- [19] J. Regehr and J. Stankovic, "HLS: a framework for composing soft real-time schedulers," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS '01)*, December 2001, pp. 3 – 14.
- [20] P. Goyal, X. Guo, and H. M. Vin, "A hierarchical CPU scheduler for multimedia operating systems," in *Proceedings of the 2nd USENIX Symposium on OS Design and Implementation (OSDI '96)*, 1996.
- [21] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," in *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS '08)*, July 2008, pp. 191 – 200.
- [22] 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.
- [23] V. Romero Segovia, "Adaptive CPU resource management for multicore platforms," Licentiate Thesis, September 2011.
- [24] M. Lindberg and K.-E. Årzen, "Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties," in *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS '10)*, December 2010, pp. 85 – 94.
- [25] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, pp. 30:1 – 30:39, April 2008.
- [26] E. Bini and G. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS '04)*, June 2004, pp. 196 – 203.
- [27] K. Esbensen, *Multivariate Data Analysis - in practice (5th Edition)*. CAMO ASA, Oslo, 2010.
- [28] N. M. Khalilzad, M. Behnam, and T. Nolte, "Adaptive hierarchical scheduling framework: Configuration and evaluation," Mälardalen University, Technical Report, April 2013.