Analysis of Hierarchical EDF Pre-emptive Scheduling

Fengxiang Zhang and Alan Burns
Real-Time Systems Research Group, Department of Computer Science,
University of York, UK
{zhangfx, burns}@cs.york.ac.uk

Abstract

This paper focuses on scheduling different hard real-time applications on a uniprocessor when the earliest deadline first algorithm is used as the local scheduler, and the global scheduler of the system could be fixed priority (FP) or earliest deadline first (EDF). Each application task could be periodic or sporadic, bound or unbound, with arbitrary relative deadline which could be less than, equal to or greater than its period. A number of different server types are considered. This paper presents an exact and efficient schedulability test for the application tasks based on the capacity demand criterion when the global scheduler could be FP or EDF, in some cases, it is necessary and sufficient. Schedulability tests which are necessary and sufficient for several types of dynamic servers are presented when the global scheduler is EDF.

1. Introduction

1.1 Hierarchical scheduling

Modern computing systems are powerful enough to execute several concurrent real-time applications on a single microprocessor. With the development of advanced high performance embedded microprocessors, hierarchical scheduling has been an active research topic in the past few years. The motivation for scheduling different hard real-time applications in a uniprocessor is cost reduction but also it offers the opportunity of performance enhancement and closer integration of distinct applications.

In hierarchical scheduling, each application is composed of a set of correlative entities (e.g. tasks or streams), each application has its own scheduler and can be executed by different scheduling algorithms, so the application can take advantage of the best features of different scheduling schemes.

Another benefit of hierarchical scheduling is that it provides temporal isolation among applications on a uniprocessor, this property prevents a misbehaving task from interfering with other tasks in another application, only the tasks in the same application as the malfunction task could be affected.

We are interested in two level hierarchical scheduling where the tasks have hard real-time constraints. Tasks in each application are running at the lower (local) level of the system, they can be pre-empted, and they are scheduled by a dispatching algorithm such as fixed priority (FP) or earliest deadline first (EDF) which is provided by the server. At the higher (global) level of the system, servers provide temporal isolation among applications, and the overall processor capacity is divided up between applications by these servers. Servers are scheduled by the global (OS) scheduler which can again be FP or EDF. Each server is assigned an execution capacity and a period of capacity replenishment; different types of servers have different capacity replenishment schemes (see Section 2.2).

1.2 EDF scheduling

The earliest deadline first (EDF) algorithm introduced by Liu and Layland [21] has been proven optimal by Dertouzos [9] among all the preemptive scheduling algorithms, in the sense that, if there exists a feasible schedule for a task set, then the schedule produced by EDF is also feasible [26]. The feasibility analysis for periodic tasks with relative deadline equal to period under EDF scheduling was first presented in [21], which showed that a set of n periodic tasks is schedulable if and only if the processor utilization is less than or equal to one.

For the task sets with arbitrary relative deadlines, there are two ways to analysis the tasks. If we just want to check the feasibility of the task set, we can use the processor demand criterion which was proposed by Baruah et al. [2, 3], and enhanced later in the literature [4, 12, 23, 28]. The worst-case response time analysis of EDF was first given by Spuri [28] with the motivation of extending the holistic theory [31] to the analysis of deadline scheduled real-time distributed systems [29]. The worst-case response time analysis given by [28] was improved later in [12].

In this paper we apply processor demand criterion, however we use the term capacity demand criterion

1052-8725/07 $25.00 © 2007 IEEE
DOI 10.1109/RTSS.2007.12

under hierarchical scheduling as the tasks in the application demand not only processing time, but also their server's capacity. In a server, if at any time there are tasks waiting to execute and the processor is idle, but the server's capacity has been exhausted, then the tasks still cannot execute.

1.3 Organization of the paper

The rest of this paper is organized as follows. Section 2 describes the system model in this paper. Section 3 describes the related work and Section 4 gives a brief introduction to the standard EDF schedulability test based on the processor demand criterion. Section 5 describes the worst-case arrival pattern of the tasks on two-level hierarchical scheduling, and gives a necessary and sufficient condition for task schedulability.

In Section 6 we address the hierarchical scheduling problem when the global scheduler is fixed priority and the local scheduler is EDF. A number of different server algorithms are considered in this section.

In Section 7, we use the dynamic periodic server (DPS), the dynamic deferrable server (DDS) or the dynamic sporadic server (DSS) [13] to construct the hierarchical system when both the global and local schedulers of the system are EDF. Section 8 gives a summary and contribution of this paper, and directions for future research.

2. Hierarchical EDF Scheduling Model

2.1 System model of this paper

A two-level hierarchical real-time system Γ comprises a number of real-time applications, and an application comprises a set of hard real-time tasks:

$\Gamma = \{A_1, A_2, ..., A_k\}$ where $A_i = \{\tau_{1_i}, \tau_{2_i}, ..., \tau_{n_i}\}$.

Task τ_i indicates the i-th task of an arbitrary application in Γ, it is characterized by a worst-case execution time C_i, a relative deadline D_i, a period T_i, and a maximum release jitter J_i.

Each task τ_i can be periodic or sporadic, and is a finite or infinite sequence of instances (requests or jobs) with a minimum inter-arrival time equal to T_i. If a job from task τ_i arrives at time a, then it will be released for execution at time $a + J_i$ and have an absolute deadline of $a + D_i$.

There is no restriction on the relationship between relative deadline D_i and period T_i of the task. Any task's relative deadline can be less than, equal to or greater than its period.

Each application is associated with a server, which has its own local scheduler to execute the tasks in the application. The local scheduler can be fixed priority or EDF. As exact analysis for fixed priority hierarchical scheduling has been presented by Davis and Burns [7], in this paper, we are only concerned with hierarchical systems in which the local scheduler is EDF.

All the tasks in each application A_i are running in the server which is associated with application A_i. So a two level hierarchical system can also be defined as a set of servers:

$$\Gamma = \{S_1, S_2, ..., S_k\},$$

where S_i indicates the i-th server in the system Γ. Each server has its own capacity C_s, server period T_s, and maximum release jitter J_s. Utilization of the server is defined as $U_s = C_s / T_s$.

Tasks in the server (application) can be bound or unbound [7]. A bound task's period is an exact multiple of its server's period, and it can only be released at the beginning of its server's period. Unbound tasks don't have any constraint on their periods and release times.

Servers are scheduled by the global (OS) scheduler which is provided by the operating system. The global scheduler can be fixed priority or EDF. If the global scheduler is fixed priority, then we should use fixed priority servers to isolate each application, otherwise, we use dynamic servers.

For space reasons, in this paper, the system model assumes that all tasks and servers are independent of each other. The existing work on fixed priority systems [8] and the conclusions obtained from the blocking analysis with EDF scheduling [5] indicates that the extension of the work presented in this paper should be straightforward.

2.2 Server algorithm

2.2.1 Fixed priority servers. In hierarchical systems when the global scheduler is fixed priority, we use the periodic server (PS) [25], the deferrable server (DS) [15, 30], or the sporadic server (SS) [27] to execute each application.

The periodic server (PS) has a fixed capacity replenishment time interval which is equal to its server period T_s. At the beginning of every period, the server capacity is replenished to its full value C_s. If there are pending tasks ready to use the server capacity and the capacity in that server period has not been exhausted, then the tasks can execute until the tasks finish or the server capacity is exhausted. During this period, the execution of the server (tasks) can be preempted by higher priority servers. If the server capacity in a period has not been exhausted but there are no tasks in the server ready to use its capacity, then the server capacity is idled away gradually with the time, as if there is always a lowest priority background task ready to use the capacity of the server.

The deferrable server (DS) has the same capacity replenishment scheme, the only difference between PS and DS is how they preserve their capacity. The DS preserves its capacity until the end of the server period if there is no pending task ready to execute. The task can execute at any time in a server period if at this time the server capacity has not been exhausted and there is no higher priority server ready to execute.

The sporadic server (SS) has a different capacity replenishment scheme. The capacity of a SS is only replenished after it has been used. The replenishment time is set T_s after the SS becomes active, and the amount of replenishment depends how much capacity it has consumed.

2.2.2 Dynamic priority servers. In this paper, three kinds of dynamic servers are considered: the dynamic periodic server (DPS), the dynamic deferrable server (DDS) and the dynamic sporadic server (DSS) [13].

The algorithms of the dynamic server is nearly the same as their fixed priority versions, the only difference between a fixed priority server and its dynamic version is the priority assignment. The dynamic server's priority is assigned by the EDF algorithm, at any time, the server which has the earliest absolute deadline will be assigned the highest priority, and the absolute deadline of a dynamic server is equal to its replenishment time.

3. Related Work

Deng et al. [10] described a two-level hierarchical preemptive scheduling model in which the global scheduler could be EDF. They used the constant utilization server to execute each application in an open system. This models each application as if it is running on a dedicated slow processor with speed equal to its server's utilization. In this model, if the tasks are preemptable, we have to know the exact arrival time of every request in the system. So this scheme is only suitable for scheduling periodic tasks or nonpreemptable tasks. In the following work of this scheduling model, Deng and Liu [11] remove this restriction by setting a scheduling quantum q that is the maximum delay time of any request permitted on the "slow processor". The effect of such a delay is equivalent to shortening every task's relative deadline by q time units. In order to make sure the tasks are still feasible, the speed of the "slow processor" most be increased which is equivalent to increasing the utilization of the server in the hierarchical system. So the system has to pay some penalty in processor utilization for nonpreemptable applications (containing sporadic or aperiodic tasks). Another drawback is that each server needs to estimate the next request occurrence time t_k for the application. The calculation of t_k has complexity $O(n)$ where n is the number of tasks, hence incurs additional scheduling overhead.

Lipari et at. [17, 18] introduced the BSS and BSS-I (we call both of them BSS for convenience) servers to support global EDF scheduling. These servers need to calculate and record the residual budget at every absolute deadline in each application, to make sure each application could not use more budget than its processor share (server utilization). Their system model has the following drawbacks: 1) The system needs to maintain a special data list for each application. 2) The data list could become very large, and the calculation and updating of the data list could be very frequent (every time a context switch occurs, or the server budget is exhausted, or there is a new arrived instance). 3) The server algorithm is not one of the standard approach usually employed with EDF systems, and it is complex to implement [20]. Following on from this system model, Lipari et at. [19] introduced the PShED algorithm which builds directly upon the BSS server algorithm, they proved that if the task set is schedulable on a slow processor with the speed equal to its application share, then it is schedulable on the full speed processor scheduled by PShED (this property for BSS was not formally proved in [17, 18]). However the PShED algorithm has all the drawbacks of the BSS server, and is more complicated to implement.

Kuo and Li [14] proposed a hierarchical scheduling scheme building upon the system architecture in [10, 11]. In this work, they used the rate monotonic (RM) [21] algorithm as the global scheduler, and the local scheduler could be RM or EDF. The sporadic server was used to execute the application. Schedulability conditions which are based on utilization test for application tasks were given. In order to get these feasibility conditions, they gave a set of strict constraints to parameters of tasks and servers. These constraints included: each task's relative deadline must be equal to the period, each server's period must be the greatest common divisor (GCD) or a divisor of the GCD of all the periods of tasks in the application, and the initial phase of each task occurs at a time point which had to be a multiple of its server period.

Saewong et al. [24] gave a multi-level hierarchical scheduling model which uses the deferrable server or the sporadic server. In each level, the deadline monotonic (DM) [16] or rate monotonic (RM) algorithm was used to schedule the reserves or tasks. In this paper, they gave a sufficient condition for feasibility based on a utilization test. They also provided worst-case response time analysis for tasks which share the same parent reserve, this analysis assumes the server capacity is only available at the end of each server period, so this worst-case response time calculation is pessimistic.

Davis and Burns [7] gave an exact analysis based on worst-case response time in the two level hierarchical system, in which both of the global and local schedulers are fixed priorities. In this work, they analyse the feasibility of hierarchical scheduling by the following steps: 1) They tested schedulability of servers by working out the worst-case response time of servers. 2) They identified the worst-case release behaviour of bound and unbound tasks in the server. 3) They tested the feasibility of tasks by calculating the worst-case response time of each task. This analysis considers the task load in a busy period, and uses this task load to calculate the exact worst-case interference due to higher priority servers in the final server period.

Lorente and Palencia [22] gave a worst-case response time analysis for tasks under two level hierarchical EDF scheduling based on the same principles as the work by Davis and Burns [7]. In this paper, they supposed each task's worst-case response time was always found in its first period when all other tasks are released simultaneously at t=0. But this is incorrect; here we give a counterexample to demonstrate this. Suppose there is a server S_i which has the highest priority in the hierarchical system, and there are three tasks τ_i, τ_j, τ_k in the server. The task we are studying is task τ_i. The parameters of the server and tasks are shown in Figure 1.

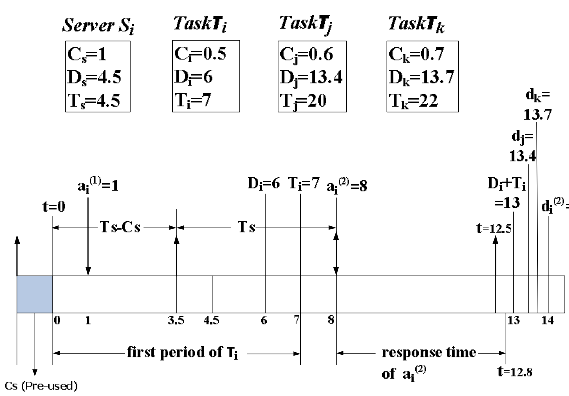

Figure 1

If all tasks except task τ_i in the server S_i are released simultaneously at t=0 when the server capacity has been exhausted as early as possible in that server period, this gives task τ_i the longest response time. The absolute deadline of the first instance of task τ_j is $d_j = 13.4$, and task τ_k is $d_k = 13.7$. The first period of task τ_i is [0,7], in this period, task τ_i has the highest priority in the server because all the other tasks' absolute deadlines in the server are larger than $D_i + T_i$, so we can easily see that the worst-case response time of task τ_i released at the first period is $T_s - C_s + C_i = 4$, this is the result predicted by Lorente and Palencia [22].

If task τ_i arrives again at $a_i^{(2)} = 8$ which is in the second period, this means that the first instance of τ_i arrives at $a_i^{(1)} = 1$ in the first period, the task τ_i will execute at t=3.5 and finish its execution by consuming capacity of S_i 0.5, then task τ_j begins to execute and consumes the remaining capacity of 0.5. The capacity of S_i replenish at t=8, and task τ_i's second instance arrives at the same time with absolute deadline $d_i^{(2)} = 14$. This time task τ_i has the lowest priority, and tasks τ_j and τ_k get to execute first. Task τ_j executes the remaining time that is 0.6-0.5=0.1, and then task τ_k begins to execute and finishes at t=8.8, and S_i's remaining capacity is 0.2. The task τ_i's second instance can only execute for 0.2 time units in this server period and has to wait for the next capacity replenishment at t=12.5, so $a_i^{(2)}$ will complete its execution at 12.5+0.3=12.8, and the response time of $a_i^{(2)}$ is 4.8, that is larger than the response time of task τ_i when it is released at any time in its first period.The results presented in this paper correctly analyse this example see Section 6.3.

4. Processor demand criterion

Baruah et al. [2, 3] presented necessary and sufficient conditions for periodic or sporadic task systems to be schedulable based on processor demand analysis. The processor demand in an interval $[0, t]$ is defined as:

$$h_R(t) = \sum_{i=1}^{n} \max\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\} C_i = \sum_{D_i \le t} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i \tag{1}$$

Lemma 1: ([3]) A task set is not feasible if and only if there exists an integer $t \ge 0$ such that $h_R(t) > t$. Furthermore, the minimum t satisfying $h_R(t) > t$ is equal to the earliest time the EDF algorithm can report failure.

Lemma 1 means a task set is schedulable by EDF if and only if in any time interval, the processor demand does not exceed the available time, that is:

$$\forall t \ge 0 \;\; h_R(t) \le t \tag{2}$$

Baruah et al. [3, 4] and Ripoll et al. [23] showed that the time interval over which equation (2) must hold is bounded:

Lemma 2: ([4, 23]) Suppose the task set is not feasible, and $\sum_{i=1}^{n} \frac{C_i}{T_i} \le 1$, then $h_R(t) > t$ implies:

$$t < \max_{1 \le i \le n}(D_i), \text{ or}$$

$$t < \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U} \le \max_{1\le i\le n}(T_i - D_i)\frac{U}{1-U},$$

where $U = \sum_{i=1}^{n}\frac{C_i}{T_i}$.

Spuri [28] gave another upper bound on the time interval needed to verify the schedulability property, that is the "as soon as possible arrival pattern" in the first processor busy period (also defined as synchronous busy period by George et al. [12]) . In this paper, he pointed out that if we remove the assumption of null release jitter, then the worst-case arrival pattern of the task set is when all tasks are released simultaneously at the beginning of the busy period t=0, then at their maximum rate, after having experienced their maximum release jitter.

Here we give a necessary and sufficient condition for EDF schedulability based on processor demand criterion, which considers the release jitter factor.

Theorem 1: A set of periodic or sporadic tasks is feasible by EDF if and only if all the following conditions are true:

1) $\sum_{i=1}^{n}\frac{C_i}{T_i} \le 1$.

2) All tasks are released simultaneously at time t=0 after having experienced their maximum release jitter, and then the subsequent instances of each task are released at their maximum rate.

3) $\forall t \in P \quad h(t) \le t$

where $h(t) = \sum_{D_i \le t + J_i}\left\lfloor \frac{t + T_i + J_i - D_i}{T_i} \right\rfloor C_i$,

J_i is the release jitter of task τ_i, P is the set of all task absolute deadlines not exceeding min(L_a, L_b), that is:

$P = \bigcup_{i=1}^{n}\{kT_i + D_i - J_i, k \in N\} \cap [0, \min(L_a, L_b)]$, where

$$L_a = \max\{(D_1 - J_1), \ldots, (D_n - J_n), \frac{\sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1-U}\},$$

and L_b is the first processor busy period when all the tasks are released simultaneously at time t=0 with the arrival pattern of condition 2, and delimited by the first processor idle time (when there is no pending task).

Proof: It can be proved by the same principles of Theorem 2's proof. See Section 5.2.

5. Conditions for schedulability

In this section we focus on the schedulability of an application that has an EDF local scheduler. We assume, in this section, that the set of servers are themselves schedulable. The global scheduler could be FP or EDF.

5.1 Worst-case arrival pattern of tasks

In hierarchical scheduling when the local scheduler is EDF, the worst-case arrival pattern of tasks is the same as the situation of fixed priority hierarchical scheduling [7]:

1) All unbound tasks in the server arrive simultaneously just after the server's capacity has been exhausted by previously arrived tasks or the "background task" if the server is a PS or a DPS, and the capacity in the server period is exhausted as soon as possible, as shown in Figure 2. This situation may occur when t_1 is the end of a hyper-period of the unbound task set in the server.
2) All bound tasks in the server arrive simultaneously at the beginning of the following server period t_2.

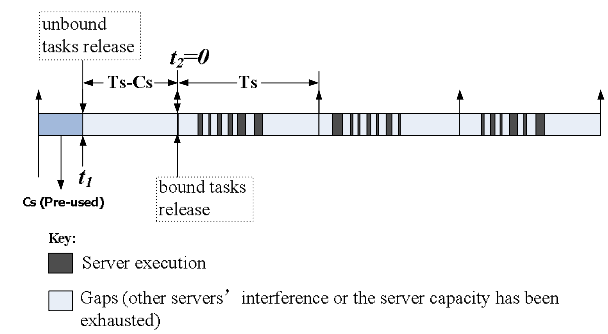

Figure 2

In this arrival pattern, if we let the release time of the bound tasks $t_2 = 0$ be the beginning of the server busy period, then each unbound task can be modeled as a bound task with a release jitter equal to $T_s - C_s$, and each bound task's release jitter equal to 0.

5.2 Conditions for schedulability of tasks

Definition 1: A server busy period is an interval of time that the studied server always has pending tasks ready to use its capacity, whether the server is executing or suspended.

Definition 2: A synchronous server busy period is a server busy period beginning at time t=0 and ended by the first server idle period (the length of such a period can be zero), when all tasks in the server are released simultaneously at time t=0 then at their maximum rate, after having first experienced their maximum release jitter.

Theorem 2: In the two level hierarchical system when the local scheduler is EDF algorithm, a set of periodic or sporadic tasks in the application is schedulable if all of the following conditions are true:

1) If all tasks are unbound: $U_{set} \le A\{T^L - (T_s - C_s)\}/T_L$;
   If there are bound tasks: $U_{set} \le U_s$;
   where U_{set} is the utilization of the task set in the application, T^L is the hyper-period of the task set, $A\{T^L - (T_s - C_s)\}$ is the worst-case available capacity of the server during $[0, T^L - (T_s - C_s)]$.
2) All tasks in the application are released simultaneously at time $t = 0$ after having experienced their maximum release jitter, and then the subsequent instances of each task released at their maximum rate. $t = 0$ is the release time of a bound task.
3) $\forall t \in P \quad R\{h(t)\} \le t$,
   where P is the set of all tasks' absolute deadlines not exceeding min(T^b, T^h), that is:
   $P = \{d_k : d_k = kT_i + D_i - J_i, d_k \le \min(T^b, T^u)\}$
   where T^b is the worst-case synchronous server busy period, and

$$T^u = \max\{(D_1 - J_1), ..., (D_n - J_n), \frac{C_s + \sum_{i=1}^{n} U_i (T_i + J_i - D_i)}{U_s - U_{set}}\}$$

$h(t)$ is the capacity demand of the tasks in the server (application) during any given time interval $[0, t]$:

$$h(t) = \sum_{D_i \le t + J_i} \left\lfloor \frac{t + T_i + J_i - D_i}{T_i} \right\rfloor C_i$$

$R\{h(t)\}$ is the worst-case response time of $h(t)$.

Proof: Condition 1 gives the maximum utilization of the task set in the application. If this utilization of the task set is larger than the requirement of Condition 1, then in a hyper-period, the total required execution time from the task set could exceed the worst-case available capacity in the server. If there's not enough capacity to deal with all requests in a hyper-period, the task set cannot be schedulable.

Condition 2 gives the worst-case arrival pattern (maximum capacity demand) of the tasks in the application. To prove this, firstly, we suppose there's no release jitter problem (all tasks in the application are bound). In each server, during any time interval $[0, \Delta]$ such that t =0 is the last time when there is no pending instance with absolute deadline less than or equal to Δ before $t = \Delta$, if we move "left" all tasks' first instances which arrive after t =0, let all tasks be released simultaneously at $t = 0$, then at their maximum rate, we can only increase the task load in the time interval $[0, \Delta]$. Then we add the release jitter consideration (there are unbound tasks). If we let the first instance of each task τ_i (arrives at time 0) be fixed, and move "left" all other instances of task τ_i in the period $[0, \Delta]$, making them closer to the first one, obviously the more we move and make the second instance closer to the first one, the task load in the period $[0, \Delta]$ could become larger. The maximum distance of each task τ_i 's other instances can be moved forward to the first one is J_i, that is the situation when all tasks arrive simultaneously at $t = 0$ after having experienced their maximum release jitter, then the subsequent instances released at their maximum rate without jitter.

Condition 3 gives a sufficient and necessary condition for the tasks schedulability. Let $A(t)$ be the worst-case available capacity of the server in any time interval $[0, t]$. $R\{h(t)\} \le t$ means there's enough capacity to deal with all $h(t)$ requests of the server before time t, so we have:

$$\begin{aligned} R\{h(t)\} \le t &\Rightarrow R\{h(t)\} \le A(t) \\ &\Rightarrow h(t) \le R\{h(t)\} \le A(t) \end{aligned} \tag{3}$$

Let $R\{A(t)\}$ be the worst-case response time of $A(t)$, then:

$$\begin{aligned} h(t) \le A(t) &\Rightarrow R\{h(t)\} \le R\{A(t)\} \\ &\Rightarrow h(t) \le R\{h(t)\} \le R\{A(t)\} \le t \end{aligned} \tag{4}$$

From formulations (3) & (4), $R\{h(t)\} \le t \Leftrightarrow h(t) \le A(t)$, and $\forall t \ge 0$, $R\{h(t)\} \le t$ is a necessary and sufficient condition of $\forall t \ge 0$, $h(t) \le A(t)$. By Lemma 1, if $\forall t \ge 0, R\{h(t)\} \le t$, the task set in the server is schedulable.

In the following part of the proof we reduce the time interval in which we need the above conditions. Since we do not change the arrival pattern of servers, the worst-case available capacity of the server does not change in any time interval. Condition 2 is the worst-case arrival pattern of the tasks in the server, following the argument by Liu and Layland [21], if there is an overflow in any tasks' arrival pattern, then there is also an overflow without idle time prior to it in the tasks' arrival pattern of Condition 2. So we can concentrate our attention only on the worst-case synchronous server busy period.

Here we give another upper bound of the time interval we need to check, when $t \ge \max\{(D_1 - J_1), ..., (D_n - J_n)\}$:

$$\begin{aligned} h(t) &= \sum_{D_i \le t + J_i} \left\lfloor \frac{t + T_i + J_i - D_i}{T_i} \right\rfloor C_i \\ &\le \sum_{i=1}^{n} \frac{t + T_i + J_i - D_i}{T_i} C_i \\ &= t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} (\frac{T_i + J_i - D_i}{T_i}) C_i \ , \end{aligned}$$

if the task set is not schedulable (we suppose the servers are schedulable), then:

$A(t) < h(t) \quad \Leftrightarrow$

$$(\left\lceil \frac{t}{T_s} \right\rceil - 1)C_s < A(t) < h(t) \le t\sum_{i=1}^{n}\frac{C_i}{T_i} + \sum_{i=1}^{n}\frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow \qquad \frac{C_s}{T_s}t - C_s < U_{set}t + \sum_{i=1}^{n}U_i(T_i + J_i - D_i)$$

$$\Leftrightarrow \qquad (U_s - U_{set})t < C_s + \sum_{i=1}^{n}U_i(T_i + J_i - D_i)$$

$$\Leftrightarrow \qquad t < \frac{C_s + \sum_{i=1}^{n}U_i(T_i + J_i - D_i)}{U_s - U_{set}}.$$

Because the task load only changes at the deadlines of the tasks, and $h(t)$ is the maximum capacity demand from the tasks in any given time interval $[0, t]$, so any task set satisfying the above conditions is schedulable □

Theorem 2 is a necessary and sufficient schedulability test for the application in which all tasks are bound, or the when its server is a periodic server or a dynamic periodic server.

For the application in which there are unbound tasks when its server is not a PS or a DPS, since assumption 1 in Section 5.1 may be pessimistic, Theorem 2 is a sufficient test in this situation.

5.3 Verifying Condition 1 of Theorem 2 for unbound tasks

For an application in which all tasks are unbound, we provide an efficient way which is sufficient to verify if $U_{set} \le A\{T^L - (T_s - C_s)\}/T^L$. If $U_{set} \le U_{\max}$, where:

$$U_{\max} = \left\lfloor \frac{T^L - (T_s - C_s)}{T_s} \right\rfloor \frac{C_s}{T^L}, \tag{5}$$

then $U_{set} \le A\{T^L - (T_s - C_s)\}/T^L$.

If $U_{set} > U_s$ then the task set is not schedulable.

If $U_{\max} < U_{set} \le U_s$, then we can calculate the worst-case response time of task load $U_{set}T^L$ in a hyper-period to know if Condition 1 is satisfied.

Lemma 3: For the application in which all tasks are unbound, Condition 1 of Theorem 2 is true if and only if $R(U_{set}T^L) \le T^L - (T_s - C_s)$, where $R(U_{set}T^L)$ is the worst-case response time of task load $U_{set}T^L$ in a hyper-period.

Proof: $R(U_{set}T_L) \le T^L - (T_s - C_s)$

$\Leftrightarrow \qquad U_{set} \cdot T^L \le A(T^L - (T_s - C_s))$

$\Leftrightarrow \qquad U_{set} \le A(T^L - (T_s - C_s))/T^L$ □

For the fixed priority servers system, $R(U_{set}T^L)$ can be computed by the method of Section 6.2, where we let $h(t) = U_{set}T^L$. For the dynamic servers system, $R(U_{set}T^L)$ can be calculated by the method of Section 7.4, and we let $h(t) = U_{set}T^L$.

To verify Condition 3 of Theorem 2, we need to know the worst-case response time of $h(t)$ at each $d_k \in P$, and the synchronous server busy period T^b for the task set. In the following sections of this paper, we will give methods for these calculations.

6. Fixed priority global scheduling

In this part, we use the periodic server (PS), the deferrable server (DS), or the sporadic server (SS) to construct the hierarchical system. The schedulability test for these fixed priority servers follows the existing approach of Davis and Burns [7], which builds on the results of maximum interference a fixed priority server may produce over lower priority servers by Bernat and Burns [6]. So in this section, we only give the schedulability test for tasks, and we assume servers are feasible.

6.1 Compute the worst-case synchronous server busy period

The worst-case synchronous server busy period T^b is a busy period when all of the tasks in the server are released simultaneously at time t=0, after having experienced their maximum release jitter, and in the last server period experience the maximum interference of other servers.

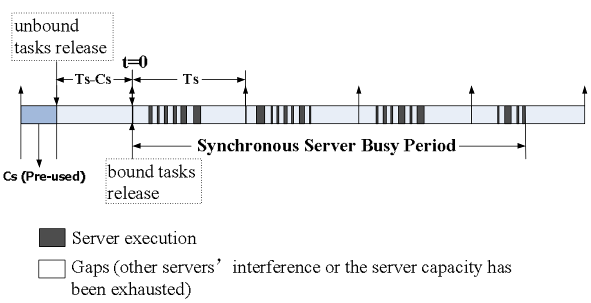

Figure 3

Here we can use the same computation method which was provided by Davis and Burns [7] to calculate T^b. The task load $L(w)$ in a server busy period w is:

$$L(w) = \sum_{i=1}^{n}\left\lceil \frac{w + J_i}{T_i} \right\rceil C_i \tag{6}$$

by Section 5.1, release jitter for each unbound task is $T_s - C_s$, and for bound task is 0.

Then the gaps in the server not including the final period in w is given by:

$$(\left\lceil \frac{L(w)}{C_s} \right\rceil - 1)(T_s - C_s) \quad (7)$$

and the busy period w extends to the final server period is:

$$w - (\left\lceil \frac{L(w)}{C_s} \right\rceil - 1)T_s \quad (8)$$

Thus the interference due to higher priority servers [6] in the final server period is:

$$\sum_{\forall X \in hp(S_i)} \left\lceil \frac{w - (\left\lceil \frac{L(w)}{C_s} \right\rceil - 1)T_s + J_x}{T_x} \right\rceil C_X \quad (9)$$

where $hp(S_i)$ is the server set in which the servers have higher priority than server S_i. J_x is the release jitter of the higher priority server S_x which belongs to $hp(S_i)$, for a periodic or sporadic server, J_x=0, and for a deferrable server, $J_x = T_x - C_x$, where T_x and C_x is the period and execution time of the server S_x.

Therefore the synchronous server busy period is:

$$w = L(w) + (\left\lceil \frac{L(w)}{C_s} \right\rceil - 1)(T_s - C_s) +$$

$$\sum_{\forall X \in hp(s)} \left\lceil \frac{w - (\left\lceil \frac{L(w)}{C_s} \right\rceil - 1)T_s + J_x}{T_x} \right\rceil C_X \quad (10)$$

The 3rd term in equation (10) is not a monotonically non-decreasing function of w, because at the beginning of the recurrent process, formula (9) may be less than zero, therefore the server busy period extends into the final period and in the equation may become an interval of negative length. So we need to modify equation (10):

$$w^{n+1} = L(w^n) + (\left\lceil \frac{L(w^n)}{C_s} \right\rceil - 1)(T_s - C_s)$$

$$+ \sum_{\forall X \in hp(s)} \left\lceil \frac{\max\{w^n - (\left\lceil \frac{L(w^n)}{C_s} \right\rceil - 1)T_s, 0\} + J_x}{T_x} \right\rceil C_X \quad (11)$$

Recurrence starts with:

$$w^0 = \sum_{i=1}^{n} C_i + (\left\lceil \frac{\sum_{i=1}^{n} C_i}{C_s} \right\rceil - 1)(T_s - C_s) \quad (12)$$

and ends with $w^{n+1} = w^n$, then we set the worst-case synchronous server busy period:

$$T^b = w^{n+1} \quad (13)$$

Lemma 4: For any application task set, if

$$R(U_{set}T^L) \le T^L - (T_s - C_s) \quad (14)$$

then equation (11) converges in a finite number of steps.

Proof: In equation (6), task load $L(w)$ is non-decreasing with w, therefore in the iterative process, $\forall k \in N$, when $w^k \le T^L - (T_s - C_s)$, we have:

$$L(w^k) \le L\{T^L - (T_s - C_s)\} = \sum_{i=1}^{n} \left\lceil \frac{T^L - (T_s - C_s) + J_i}{T_i} \right\rceil C_i$$

$$\le \sum_{i=1}^{n} \left\lceil \frac{T^L}{T_i} \right\rceil C_i = \sum_{i=1}^{n} \frac{T^L}{T_i} C_i = \sum_{i=1}^{n} \frac{C_i}{T_i} T^L = U_{set} T^L .$$

Let $R\{L(w)\}$ be the worst-case response time of $L(w)$, then w^k in equation (11), $w^k \le R\{L(w^{k-1})\}$, therefore when $L(w^{k-1}) \le U_{set}T^L$, we have:

$$w^k \le R\{L(w^{k-1})\} \le R(U_{set}T^L) \le T^L - (T_s - C_s) .$$

From above argument, if w^k continuously increases with the recurrence (the value of k also increases), $L(w^k)$ will get to the value $L(w^k) = U_{set}T^L$. Then:

$$w^{k+1} \le R\{L(w^k)\} = R(U_{set}T^L) \le T^L - (T_s - C_s)$$

so $\forall m > k$, $m \in N$,

$$L(w^m) = U_{set}T^L \text{ and } w^m \le R(U_{set}T^L)$$

therefore there exists $n > k$, $n \in N$,

$$w^{n+1} = R(U_{set}T^L) = w^n \quad \square$$

Lemma 4's condition can be verified by the same method of Section 5.3. Here the task set can be of any kind and it may contain bound tasks.

If Lemma 4's condition is unsatisfied, then we only use T^u as the upper bound of the deadlines that we have to check.

6.2 Compute the worst-case response time of capacity demand h(t)

Because

$$h(t) = \sum_{D_i \le t + J_i} \left\lfloor \frac{t + T_i + J_i - D_i}{T_i} \right\rfloor C_i \quad (15)$$

is a determined value when t is fixed, we do not need to consider any new incoming task load during the response time of $h(t)$. So we can assume that there is a task with the highest priority in the server, its execution time equal to $h(t)$, and only calculate the worst-case response time of this task in the last server period (the period $[t_f, t_h]$ in Figure 4), then we can get the whole response time of $h(t)$ (the $R\{h(t)\}$ term).

Let $l(t)$ be the task load of $h(t)$ that extends into the last server period, then $l(t)$ is:

$$l(t) = h(t) - (\left\lceil \frac{h(t)}{C_s} \right\rceil - 1)C_s \tag{16}$$

$R\{l(t)\}$ is calculated by:

$$w^{n+1} = l(t) + \sum_{\forall X \in hp(s)} \left\lceil \frac{w^n + J_x}{T_x} \right\rceil C_x \tag{17}$$

where $hp(S_i)$ is the server set in which the servers have higher priority than server S_i. J_x is the release jitter of the higher priority server S_x which belongs to $hp(S_i)$, for a periodic or sporadic server, J_x=0, and for a deferrable server, $J_x = T_x - C_x$. Recurrence starts with:

$$w^0 = l(t), \tag{18}$$

and ends with $w^{n+1} = w^n$ (we assume that the servers are schedulable). Then we set: $R\{l(t)\} = w^{n+1}$. So the worst-case response time of $h(t)$ is:

$$R\{h(t)\} = (\left\lceil \frac{h(t)}{C_s} \right\rceil - 1)T_s + R\{l(t)\} \tag{19}$$

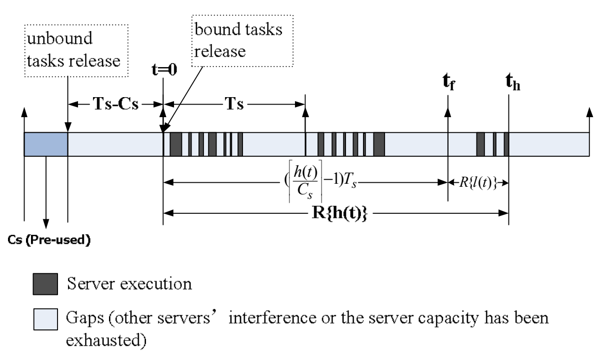

Figure 4

6.3 Illustrations of the analysis

Example A: Here we revisit the example given in Section 3. The parameters of the server and tasks are shown in Figure 1. The position of $t = 0$ (different from Figure 1) and release time of the tasks are shown in Figure 4. We assume that all tasks are unbound in this example. As we mentioned before, we only test the schedulability of the tasks in this section.

Step 1 (Verify Condition 1 of Theorem 2):

$$U_{set} = \frac{0.5}{7} + \frac{0.6}{20} + \frac{0.7}{22} \cong 0.1332$$

$$U_{\max} = \left\lfloor \frac{140 - 3.5}{4.5} \right\rfloor \frac{1}{140} \cong 0.2143$$

As $U_{set} \le U_{\max}$, then Condition 1 of Theorem 2 is satisfied.

Step 2 (Calculate synchronous server busy period T^b):

$$w^0 = 1.8 + (\left\lceil \frac{1.8}{1} \right\rceil - 1)(4.5 - 1) = 5.3$$

$$L(w^0) = \left\lceil \frac{5.3+3.5}{7} \right\rceil 0.5 + \left\lceil \frac{5.3+3.5}{20} \right\rceil 0.6 + \left\lceil \frac{5.3+3.5}{22} \right\rceil 0.7 = 2.3$$

$$w^1 = 2.3 + (\left\lceil \frac{2.3}{1} \right\rceil - 1)(4.5 - 1) + 0 = 9.3$$

$$L(w^1) = \left\lceil \frac{9.3+3.5}{7} \right\rceil 0.5 + \left\lceil \frac{9.3+3.5}{20} \right\rceil 0.6 + \left\lceil \frac{9.3+3.5}{22} \right\rceil 0.7 = 2.3$$

$$w^2 = 2.3 + (\left\lceil \frac{2.3}{1} \right\rceil - 1)(4.5 - 1) + 0 = 9.3$$

As $w^2 = w^1$, the synchronous server busy period T^b=9.3.

Step 3 (Calculate another upper bound of the time interval T^u):

$$T^u = \frac{1 + (\frac{0.5}{7} \times 4.5 + \frac{0.6}{20} \times 10.1 + \frac{0.7}{22} \times 11.8)}{1/4.5 - (0.5/7 + 0.6/20 + 0.7/22)} \cong 22.4768$$

Because $T^b < T^u$, the tasks' absolute deadlines we need to check are:

$$\{d_k : d_k = kT_i + D_i - J_i, d_k \le T^b = 9.3\}$$

Step 4 (Verify Condition 3 of Theorem 2):
There's only one absolute deadline in the period [0,9.3], that is $d_k = 2.5$,

when t=2.5, $h(t) = \left\lfloor \frac{2.5 + 7 + 3.5 - 6}{7} \right\rfloor 0.5 = 0.5$,

according to equations (16)(17)(18)(19), $R(h(t)) = 0.5$ (remember $t = 0$ is the release time of a bound task, and the server has the highest priority and hence suffers no interference), so $R(h(t)) \le t = 2.5$. Hence all conditions in Theorem 2 are satisfied, and the task set is schedulable.

Example B: In order to give a further illustration of how this mechanism works, we change the parameters of the tasks in Example A. If we let each of these three tasks' execution time be 0.1, it is easy to see $T^b = 0.3$. Since there's no absolute deadline in the period $[0, 0.3]$, we don't need to check any absolute deadline. This means, from the tasks point of view, any busy period of the tasks (obviously T^b is the longest one) finishes before it meets a deadline, so such a task set is always schedulable.

Example C: In this example, we let the parameters of the tasks and server S_i be the same as Example A, and S_i be a periodic server. If there is another higher priority server with period $T_s = 4.5$ and execution time $C_s = 1$, then server S_i could get the maximum interference of 1 time unit in the final server period of T^b, and so $T^b = 10.3$. Therefore there are three absolute deadlines we need to check: $\{d_1 = 2.5, \ d_2 = 9.9, \ d_3 = 10.2\}$. When $t = d_3 = 10.2$, $h(t) = 2.3$, $R\{(h(t))\}$ = 10.3. Because $R\{h(t)\} > t$, the task set is unschedulable.

What these three examples illustrate is that while the derivation of the test may be considered complex, its application is straightforward and easily incorporated into tools.

7. EDF Global Scheduling

In this part, we use the dynamic periodic server (DPS), the dynamic deferrable server (DDS) or the dynamic sporadic server (DSS) to construct the hierarchical system, thus both of the global and local schedulers are EDF. We will present the schedulability tests for these different server types, and the schedulability test for tasks based on Theorem 2.

7.1 Maximum interference of a dynamic server

7.1.1 Dynamic periodic server. The DPS is time-triggered and can be regarded as an ordinary periodic task with execution time C_s and with release jitter equal to zero. So the remaining objective of this section is to find out the maximum interference a DDS or DSS can produce over other servers.

7.1.2 Dynamic deferrable server.

Lemma 5 ([13]) For every time interval $[t', t]$ such that t is a missed deadline, $t' \le t - C_s$, and there is no task with deadline later than t executing in the interval, an upper bound of the dynamic deferrable server's demand for CPU time in the interval is:

$$\left\lfloor \frac{\Delta - C_s}{T_s} \right\rfloor C_s + C_s ,$$

where $\Delta = t - t'$ is the length of the interval.

Lemma 5 tells us that in any time interval $[0, \Delta]$, the worst-case (maximum) interference a dynamic deferrable server can produce over other servers occurs when: 1) There is a request with execution time $C_i \ge C_s$ in the server, 2) and this request arrives at C_s time units before the end of that server period, let the arrival time be t', t' should be the beginning time of the interval $[0, \Delta]$, 3) and there should be no other request arrival before t' in that server period, 4) and in every following server period the server's capacity can be exhausted. As shown in Figure 5.

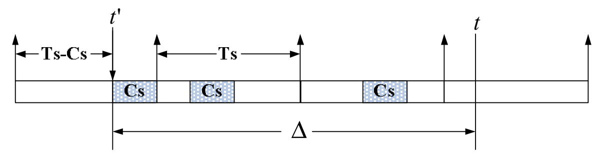

Figure 5

So we can conclude that when a DDS produces maximum interference over other servers, the DDS can be modeled as a periodic task with execution time equal to C_s, relative deadline equal to its period, and release jitter equal to $T_s - C_s$.

7.1.3 Dynamic sporadic server.

Lemma 6 ([13]) For every time interval $[t', t]$ such that t is a missed deadline, $t' \le t - C_s$, and there is no task with deadline later than t executing in the interval, an upper bound of the dynamic sporadic server's demand for CPU time in the interval is:

$$\left\lfloor \frac{\Delta}{T_s} \right\rfloor C_s$$

where $\Delta = t - t'$ is the length of the interval.

From Lemma 6, in the worst-case, we can regard the DSS as a periodic task with execution time equal to C_s, relative deadline equal to its period, and release jitter equal to 0.

7.2 Schedulability of servers

Following Section 7.1, each dynamic server can be modeled as a periodic task with execution time equal to its capacity C_s, relative deadline equal to its period T_s, and release jitter depends on the server type.

As each dynamic polling server (DPS) or dynamic sporadic server (DSS) can be regarded as a periodic task without release jitter, so if there are only DPS and DSS in a hierarchical system, we can use the sufficient and necessary condition by Liu and Layland [21] to test the feasibility of servers, that is:

If $$\sum_{i=1}^{n} U_{Si} \le 1 ,$$

then servers are schedulable, where U_{Si} is the utilization of server i, equal to C_s / T_s.

If there are dynamic deferrable servers (DDS) in the system, we can use Theorem 1 to test the servers feasibility. For a DDS, the release jitter $J_i = T_s - C_s$, for a DPS or DSS, $J_i = 0$.

For the tasks in each application, we still use Theorem 2 to test their schedulability.

7.3 Compute the worst-case synchronous server busy period

When we are calculating the worst-case synchronous server busy period T^b in this section, in order to reduce the calculation complexity, we suppose the server capacity is only available at the end of the final server period, as shown in Figure 6. Note that in this assumption, we don't increase the pessimism for schedulability, we only increase the length of T^b, so we may need to check more absolute deadlines when we are verifying the feasibility of the tasks, but we save a lot of computation time in calculating the exact value for T^b in the EDF global scheduling environment.

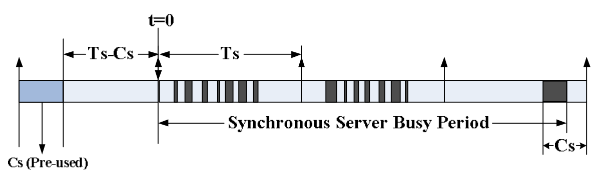

Figure 6

The task load in the synchronous server busy period w is:

$$L_i(w) = \sum_{i=1}^{n} \left\lceil \frac{w + J_i}{T_i} \right\rceil C_i \tag{20}$$

release jitter for each unbound task is $T_s - C_s$, and for each bound task is 0.

The total length of the worst-case synchronous server busy period is:

$$w^{n+1} = L(w^n) + \left\lceil \frac{L(w^n)}{C_s} \right\rceil (T_s - C_s) \tag{21}$$

Recurrence starts with:

$$w^0 = \sum_{i=1}^{n} C_i + \left\lceil \frac{\sum_{i=1}^{n} C_i}{C_s} \right\rceil (T_s - C_s) \tag{22}$$

and ends with $w^{n+1} = w^n$, then we set the worst-case synchronous server busy period:

$$T^b = w^{n+1} \tag{23}$$

In order to ensure equation (21) converges, we need to make sure utilization of the task set $U_{set} \le U_{\max}$ before we calculate T^b, where

$$U_{\max} = \left\lfloor \frac{T^L - (T_s - C_s)}{T_s} \right\rfloor \frac{C_s}{T^L} \tag{24}$$

If $U_{set} > U_{\max}$, then we only use T^u as the upper bound of the deadlines we have to check.

7.4 Compute the worst-case response time of h(t)

The difference in the response time of $h(t)$ between fixed priority servers and dynamic servers is only in the length of the last server period (the period $[t_f, t_h]$ in Figure 4). It is easy to see that before t_f, the length of $h(t)$'s worst-case response time is the same, no matter what kind of server is being used.

The task load of $h(t)$ in the last server period $l(t)$ can be calculated by equation (16).

The task load $l(t)$ can be regarded as a periodic task with execution time equal to $l(t)$, relative deadline equal to its server period, and release jitter equal to zero. Following Section 7.1, all other servers in the system are modeled as periodic tasks with execution time equal to their capacity, relative deadlines equal to their periods, and for a dynamic polling and sporadic server, release jitter equal to zero, for a dynamic deferrable servers, release jitter equal to $T_s - C_s$. Then the worst-case response time of $l(t)$ can be calculated by the existing EDF response time analysis by George et al [12].

Let $R\{l(t)\}$ be the worst-case response time of $l(t)$. After obtaining $R\{l(t)\}$ by the method of [12], the worst-case response time of $h(t)$ is:

$$R\{h(t)\} = (\left\lceil \frac{h(t)}{C_s} \right\rceil - 1) T_s + R\{l(t)\}. \tag{25}$$

8. Conclusions

In this paper, we present schedulability analysis for two-level hierarchical scheduling when the local scheduler of the system is EDF, and the global scheduler could be FP or EDF. A schedulability test which considers the release jitter problem for tasks on standard EDF scheduling is also given, and it can be used for testing the feasibility of the dynamic servers considered in this work. We point out a mistake in the literature which miss calculates each task's worst-case response time for hierarchical scheduling when the local scheduler is EDF.

We present an exact and efficient schedulability test for application tasks based on the capacity demand criterion when the local scheduler is EDF and the global scheduler is FP or EDF. Each application task could be periodic or sporadic, bound or unbound, with arbitrary relative deadline which could be less than, equal to or greater than its period. When the studied application's server is a PS or a DPS, or the application consists of only bound tasks, the schedulability test is necessary and sufficient. A number of server types are considered corresponding to various global scheduling algorithms. We evaluate the maximum interference a dynamic server may produce over other servers, and present sufficient and necessary feasibility tests for several types of dynamic servers when the global scheduler is EDF.

As future work, we plan to extend this system model to share local resources between the tasks, and global resources between the applications, and we are going to consider some other server types, i.e. the constant bandwidth server [1]. Making experimental comparisons between fixed priority and EDF hierarchical scheduling is also an interesting area for future work. Optimal server parameters selection is another direction of research.

9. References

[1] L. Abeni and G. Buttazzo. "Integrating Multimedia Applications in Hard Real-Time Systems". In *Proceedings 19th IEEE Real-Time Systems Symposium*, pp.4-13, 1998.

[2] S.K. Baruah, L.E. Rosier and R.R. Howell. "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on one Processor". *Real-Time Systems*, 2(4):301-324,1990.

[3] S.K. Baruah, A.K. Mok and L.E. Rosier. "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In *Proceedings 11th IEEE Real-Time System Symposium*, pp.182-190,1990.

[4] S.K. Baruah, R.R. Howell, and L.E. Rosier. "Feasibility Problems for Recurring Tasks on One Processor". *Journal of Real-Time Systems*, Vol.118, pp.3-20, 1993.

[5] S.K. Baruah. "Resource Sharing in EDF-Scheduled Systems: A Closer Look". In *Proceedings 27th IEEE Real-Time Systems Symposium*, pp.379-387, 2006.

[6] G. Bernat and A. Burns. "New Results on Fixed Priority Aperiodic Servers". In *Proceedings 20th IEEE Real-Time Systems Symposium*, pp.68-78, 1999.

[7] R.I. Davis and A. Burns. "Hierarchical Fixed Priority Preempive Scheduling". In *Proceedings 26th IEEE Real-Time Systems Symposium*, pp.389-398, 2005.

[8] R.I. Davis and A. Burns. "Resource Sharing in Hierarchical Fixed Priority Pre-emptive Systems". In *Proceedings 27th IEEE Real-Time Systems Symposium*, pp.257-270, 2006.

[9] M.L. Dertouzos. "Control Robotics: The Procedural Control of Physical Processes". In *Proceedings of the IFIP congress*, pp.807-813, 1974.

[10] Z. Deng, J.W.-S. Liu, and J. Sun. "A Scheme for Scheduling Hard Real-Time Applications in Open System Environment". In *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pp.191-199, June, 1997.

[11] Z. Deng and J.W.-S. Liu. "Scheduling Real-Time Applications in an Open Environment". In *Proceedings 18th IEEE Real-Time Systems Symposium*, pp308-319, December 1997.

[12] L. George, N. Rivierre, and M. Spuri. "Preemptive and non-preemptive Real-Time uniprocessor Scheduling". *Technical Report 2966*, INRIA, France, 1996.

[13] T.M. Ghazalie and T.P. Baker. "Aperiodic Servers in a Deadline Scheduling Environment". *Real-Time Systems*, 9(1): 31-67, 1995.

[14] T-W. Kuo and C-H. Li. "A Fixed Priority Driven Open Environment for Real-Time Applications". In *Proceedings 20th IEEE Real-Time Systems Symposium*, pp.256-267, 1999.

[15] J.P. Lehoczky, L. Sha, and J.K. Strosnider. "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments". In *Proceedings 8th IEEE Real-Time System Symposium*, pp.261-270, 1987.

[16] J. Leung and J.W. Whitehead. "On the Complexity of Fixed Priority Scheduling of Periodic Real-Time Tasks". *Performance Evaluation*, 2(4), 1982.

[17] G. Lipari and G. Buttazzo. "Scheduling Real-Time Multi-Task Applications in an Open System". In *Proceedings 11th Euromicro Conference on Real-Time Systems*, 1999.

[18] G. Lipari and S. Baruah. "Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems". In *Proceedings of the Real-Time Technology and Applications Symposium*, pp166-175, 2000.

[19] G. Lipari, J. Carpenter, and S. Baruah. "A Framework for Achieving Inter-application Isolation in Multiprogrammed, Hard Real-Time Environments". In *Proceedings 21th IEEE Real-Time Systems Symposium*, pp.217-216, 2000.

[20] G. Lipari and E. Bini. "Resource Partitioning among Real-Time Applications". In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pp.151-158, 2003.

[21] C.L. Liu and J.W. Layland. "Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment". *Journal of ACM* 20(1):40-61, January 1973.

[22] J.L. Lorente and J.C. Palencia. "An EDF Hierarchical Scheduling Model for Bandwidth Servers". In *Proceedings of 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. pp.261-266, 2006.

[23] I. Ripoll, A. Crespo and A.K. Mok. "Improvement in Feasibility Testing for Real-Time Tasks". *Journal of Real-Time Systems*, Vol.11, pp.19-39, 1996.

[24]S. Saewong, R. Rajkumar, J.P. Lehoczky, and M.H. Klein. "Analysis of Hierarchical Fixed-Priority Scheduling". In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pp.173, 2002.

[25] L. Sha, J.P. Lehoczky, and R. Rajkumar. "Solutions for some Practical Problems in Prioritised Preemptive Scheduling". In *Proceedings of the IEEE Real-Time Systems Symposium*. pp.181-191,1986.

[26]L. Sha, T. Abdelzaher, K.E. Årzén, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, A. Cervin, J.P. Lehoczky and A.K. Mok. "Real time Scheduling Theory: A Historical Perspective". *Real-Time Systems*, 28(2-3):101–155, 2004.

[27] B. Sprunt. "Aperiodic Task Scheduling for Real-Time Systems". *Ph.D. Dissertation*, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1990.

[28] M.Spuri. "Analysis of Deadline Schedule Real-time Systems". *Technical Report 2772*, Inria, France, 1996.

[29] M. Spuri. "Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems". *Technical Report 2873*, INRIA, France, 1996.

[30] J.K. Strosnider, J.P. Lehoczky, and L. Sha. "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments". *IEEE Transactions on Computers*, 44(1):73-91, January 1995.

[31] K.W. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". *Microprocessors and Microprogramming* 40, 1994.