Hierarchical Fixed Priority Pre-emptive Scheduling

R.I.Davis and A.Burns
Real-Time Systems Research Group, Department of Computer Science,
University of York, YO10 5DD, York (UK)
rob.davis@cs.york.ac.uk, alan.burns@cs.york.ac.uk

Abstract

This paper focuses on the hierarchical scheduling of systems where a number of separate applications reside on a single processor. It addresses the particular case where fixed priority pre-emptive scheduling is used at both global and local levels, with a server associated with each application. Using response time analysis, an exact schedulability test is derived for application tasks. This test improves on previously published work. The analysis is extended to the case of harmonic tasks that can be bound to the release of their server. These tasks exhibit improved schedulability indicating that it is advantageous to choose server periods that enable some tasks to be bound to the release of their server. The use of Periodic, Sporadic and Deferrable Servers is considered with the conclusion that the simple Periodic Server dominates both Sporadic and Deferrable Servers when the metric is application task schedulability.

1. Introduction

In automotive electronics, the advent of advanced high performance embedded microprocessors such as Freescale Semiconductor's (Motorola's) MPC5200 (PowerPC 603), Infineon's TC1765 (TriCore) and NEC's V850E/RS1 have made possible functionality such as adaptive cruise control, lane departure warning systems, integrated telematics and satellite navigation applications as well as advances in engine management, transmission control and body electronics. Where low-cost 8 and 16-bit microprocessors were previously used as the basis for separate Electronic Control Units (ECUs) each supporting a single hard real-time application, there is now a trend towards integrating functionality into a smaller number of more powerful microprocessors. The motivation for such integration comes mainly from cost reduction but also offers the opportunity of functionality enhancement. This trend in automotive electronics is mirrored by a similar trend in avionics.

Integrating a number of real-time applications onto a single microprocessor raises issues of resource allocation and partitioning. Disparate applications require access to processor and other resources in a manner that ensures they are able to complete the necessary computations within specified time constraints, whilst ensuring that they do not impinge upon the real-time behaviour of other applications.

When composing a system comprising a number of applications, it is typically a requirement to provide temporal isolation between the various applications. This enables the properties of previous system designs, where each application resided on a separate processor, to be retained. In particular if one application fails to meet its time constraints then there should be no knock on effects on other unrelated applications. There is currently considerable interest in hierarchical scheduling as a way of providing temporal isolation between applications executing on a single processor.

In a hierarchical system, a *global* scheduler is used to determine which application should be allocated the processor at any given time and a *local* scheduler is used to determine which of the chosen application's tasks should actually execute. A number of different scheduling schemes have been proposed for both global and local scheduling. These include cyclic or time slicing frameworks, dynamic priority based scheduling and fixed priority scheduling. In this paper we focus on the use of fixed priority pre-emptive scheduling (FPPS) for both global and local scheduling.

Fixed priority pre-emptive scheduling offers advantages of flexibility over cyclic approaches whilst being sufficiently simple to implement; that it is possible to construct highly efficient embedded real-time operating systems that use this scheduling policy.

The basic framework for a system utilising hierarchical fixed priority pre-emptive scheduling is as follows. The system comprises a number of applications each of which is composed of a set of

tasks. A separate *server* is allocated to each application. Each server has an execution capacity and a replenishment period, enabling the overall processor capacity to be divided up between the different applications. Each server has a unique priority that is used by the global scheduler to determine which of the servers with capacity remaining and tasks ready to execute should be allocated the processor. Further, each task has a unique priority within its application. The local scheduler, within each server, uses task priorities to determine which of an application's tasks should execute when the server is active. The basic model assumes that tasks and applications are independent, however the model can be extended to allow local resource sharing between tasks in the same application and global resource sharing between tasks in different applications.

1.1. Related work

Kuo and Li [1] first introduced analysis of hierarchical fixed priority pre-emptive scheduling, building upon the work of Deng and Liu [2]. The analysis provided by Kuo and Li considered the use of Sporadic Servers [3] to execute applications. Using the techniques of Liu and Layland [4] they provided a simple utilisation based schedulability test. However for this utilisation based test to be applicable, severe restrictions were placed on the server parameters. In particular, each server period had to be the greatest common divisor (GCD) or a divisor of the GCD of all the tasks in the application.

Saewong et al [5] provided response time analysis for hierarchical systems using Deferrable Servers [6] or Sporadic Servers [3] to schedule a set of hard real-time applications. This analysis assumes that in the worst-case a server's capacity is made available at the end of its period. Whilst this is a safe assumption it is also pessimistic, for example, the highest priority server will typically have a response time that is much shorter than its period and so will always be able to make capacity available earlier than considered in [5]. The schedulability analysis given in [5] is sufficient but not necessary: there are some systems that it would deem unschedulable that are in fact schedulable.

Lipari and Bini [7] provide an alternative response time analysis formulation using an availability function to represent the time made available by a server from an arbitrary time origin. This formulation again makes the assumption that in the worst-case; server capacity is made available at the very end of the server's period. Lipari and Bini also investigate the problem of server parameter selection and consider choice of replenishment period and capacity for a single server in isolation, using a geometric approach based on an approximation of the server availability function.

In [8], Almeida builds upon the work of Lipari and Bini. The analysis given in [8] recognises that the server availability function depends on the *"maximum jitter that periods of server availability may suffer"*. A parameter (delta) is introduced into the analysis to represent the initial latency in server capacity becoming available. We note that setting this parameter to reflect the server's computed worst-case response time and hence its maximum jitter would result in more accurate analysis as demonstrated in section 3 of this paper.

1.2. Organisation

Section 2 describes the terminology, notation and system model used in the rest of the paper.

Section 3 presents schedulability analysis tests that compute the *exact* worst-case response time of tasks scheduled under a set of Servers. This analysis is extended to accurately model *bound* tasks, the releases of which are synchronised with their server's period. The analysis can also be extended to account for access to global shared resources.

In section 4, we evaluate the exact analysis presented in the previous section by comparing its effectiveness to that of previously published schedulability tests. Our evaluation investigates the effects of server context switch overheads and server algorithm selection on system schedulability.

Section 5 summarises the major contributions of the paper and suggests directions for future research.

2. Hierarchical scheduling model

2.1. Terminology and system model

We are interested in the problem of scheduling multiple real-time *applications* on a single processor. Each application comprises a number of real-time *tasks*. Associated with each application is a *server*. The application tasks execute within the associated server, which affords them temporal isolation.

Scheduling takes place at two levels: *global* and *local*. The global scheduling policy determines which server has access to the processor at any given time, whilst the local scheduling policy determines which application task that server should execute. In this paper we analyse systems where the fixed priority pre-emptive scheduling policy is used for both global and local scheduling.

Application tasks may arrive and become ready to execute either *periodically* at fixed intervals of time, or *sporadically* after some minimum inter-arrival time has elapsed. Each application task τ_i, has a unique priority i within its application and is characterised by its

relative *deadline* D_i, *worst-case execution time* C_i, and minimum inter-arrival time T_i, otherwise referred to as its *period*. In addition, we will assume that each application contains one or more soft real-time tasks. These soft tasks are assumed to execute at lower priorities than the hard real-time tasks. The soft real-time tasks may however consume server capacity and hence affect the worst-case scenario for hard real-time task execution.

Each server has a unique priority *s*, within the set of servers and is characterised by its *capacity* C_S, *replenishment period* T_S, and *jitter* J_S. A server's capacity is the maximum amount of execution time that may be consumed by the server in a single invocation. The replenishment period is the minimum time before the server's capacity is available again. The server's jitter is the difference between the minimum and maximum time that can elapse between replenishment of the server's capacity and that capacity starting to be consumed given no higher priority interference.

Application tasks are referred to as *bound* or *unbound* [11]. Bound tasks have a period that is an exact multiple of their server's period and arrival times that coincide with replenishment of the server's capacity. Thus bound tasks are only ever released at the same time as their server. All other tasks are referred to as unbound.

A task's worst-case response time R_i, is the longest possible time from the task arriving to it completing execution. Similarly, a server's worst-case response time R_S, is the longest possible time from the server being replenished to its capacity being exhausted, given that there are tasks ready to use all of the server's capacity. A task is said to be *schedulable* if its worst-case response time does not exceed its deadline. A server is schedulable if its response time does not exceed its period.

The *critical instant* [4] for a task is defined as the pattern of execution of other tasks and servers that leads to the task's worst-case response time.

The analysis presented in this paper assumes that all applications and tasks are independent. We have lifted this restriction and extended the analysis to take account of blocking effects due to tasks accessing resources that are either shared locally within a single application or globally between tasks in different applications. Space restrictions prevent us from describing our work in this area here. Full details are however available in a technical report [13].

2.2. Servers

In this paper we consider the Deferrable Server (DS) [6] the Sporadic Server (SS) [3] and the Periodic or Polling Server (PS) [12].

The *Periodic Server* is invoked with a fixed period. If there are application tasks ready to use the server's capacity, then they are executed until the tasks either complete or the server's capacity is exhausted. If there are no tasks ready to use the server then its capacity is assumed to be idled away, as if there was a background task that is always ready to execute. Once the server's capacity is exhausted, the server suspends execution until its capacity is replenished at the start of its next period. If a task arrives before the server's capacity has been exhausted then it will be serviced. Execution of the server may be delayed and or pre-empted by the execution of other servers of a higher priority.

The *Deferrable Server* is also invoked with a fixed period. It differs from the Periodic Server in that if no tasks are ready to use the server then it may suspend its execution, preserving its capacity. The Deferrable Server's capacity may be preserved throughout its period. If an application task becomes ready late in the server's period it can be executed until either the server's capacity is exhausted or the end of the server's period is reached. At the end of the server's period any remaining server capacity is discarded and the server's capacity is then replenished. Again execution of the server may be delayed and or pre-empted by the execution of other servers of a higher priority. Schedulability analysis of the Deferrable Server needs to take account of the well-known phenomenon of *back-to-back* hits. By preserving its capacity until near the end of its period a high priority Deferrable Server can cause back-to-back interference of $2C_S$ on lower priority servers. Effectively a Deferrable Server has a jitter equal to $T_S - C_S$ [9].

The *Sporadic Server* differs from both the Periodic Server and the Deferrable Server in that its capacity is only replenished after it has been used. In [3], Sprunt proved that in the worst-case the interference due to a Sporadic Server is equivalent to that of a simple Periodic Server. The implementation complexity and overheads of the Sporadic Server are however significantly greater than those of either the Periodic or Deferrable Server due to the requirement to keep track of a number of different replenishment times and capacities.

2.3. Busy periods and loads

The analysis presented in section 3 makes use of the concepts of *busy periods* and *loads*. For a particular application, a priority level *i* busy period is defined as an interval of time during which there is outstanding task execution at priority level *i* or above. Busy periods may be represented as a function of the outstanding execution time at and above a given priority level, thus $w_i(L)$ is used to represent a priority level *i* busy period

(or 'window', hence *w*) equivalent to the longest time that the application's server can take to execute a given load *L*.

The load on a server is itself a function of the time interval considered. We use $L_i(w)$ to represent the total task executions of priority level *i* and above, released within a time window of length *w*.

3. Schedulability analysis

In this section we present exact schedulability analysis for applications comprising bound and unbound hard real-time tasks executing under a set of servers in a hierarchical system scheduled at both global and local levels according to the fixed priority pre-emptive scheduling policy.

3.1. Exact analysis

We derive the exact worst-case response time for a task τ_i, executing under a server *S*, using the principles of Response Time Analysis [10] as follows:

1. Determine the *critical instant*: the pattern of server and task execution that leads to the worst-case response time of the task.
2. Derive a formula for $L_i(w)$, the load at priority level *i* and above, released in a window of length *w* starting at the critical instant.
3. Derive a formula for $w_i(L)$, the length of the priority level *i* busy period starting at the critical instant and finishing when the server has completed execution of the load *L*.
4. Combine the formulae for $L_i(w)$ and $w_i(L)$ into a recurrence relation that can be solved to find the worst-case response time of task τ_i.

The critical instant for a task scheduled under a server occurs when:

1. The server's capacity has been exhausted by lower priority tasks as early in its period as possible.
2. The task of interest (if its is *unbound*) and all higher priority unbound tasks in the application arrive just after the server's capacity has been exhausted.
3. The task of interest (if its is *bound*) and all higher priority *bound* tasks in the application arrive at the start of the server's next period.
4. The server's capacity is replenished at the start of its next period, however further execution of the server is then delayed for as long as possible due to interference from higher priority servers.

Figure 1 illustrates the worst-case response times for a task depending on whether it is bound or unbound.

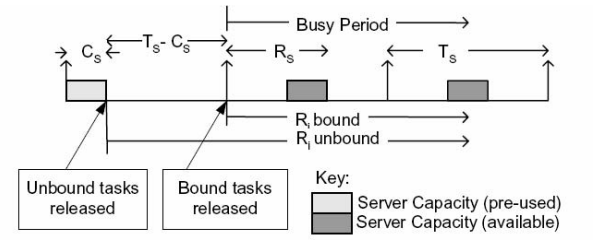

Figure 1 Critical instant

We can determine the worst-case response time of a task τ_i by computing the length of the priority level *i* busy period starting at the first release of the server that could execute the task (see Figure 1). This busy period can be viewed as being made up of three components:

1. The execution of task τ_i and tasks of higher priority released during the busy period.
2. The gaps in any complete periods of the server.
3. Interference from higher priority servers in the final server period that completes the execution of task τ_i.

Figure 2 illustrates the busy period in more detail.

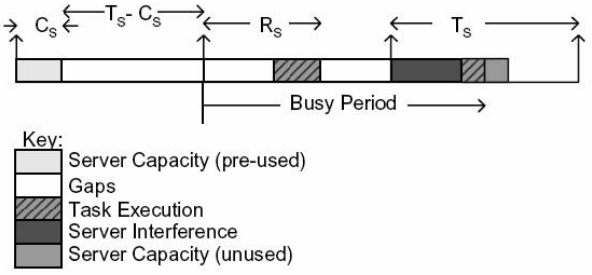

Figure 2 Busy Period

The task load at priority level *i* and higher, to be executed in the busy period *w*, is given by:

$$L_i(w) = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w + J_i}{T_j} \right\rceil C_j \qquad (1)$$

Where *hp(i)* is the sets of tasks that have priorities higher than task τ_i and J_i is the release jitter of the task relative to the release of the server. This is zero for a bound task and $(T_S - C_S)$ for an unbound task.

The total length of gaps in complete server periods, not including the final server period, is given by:

$$\left(\left\lceil \frac{L_i(w)}{C_S} \right\rceil - 1 \right) (T_S - C_S) \qquad (2)$$

The interference due to higher priority servers executing during the final server period that completes execution of task τ_i can be modeled in a number of ways.

1. The analysis given by Saewong et al in [5]

assumes that each server's worst-case response time is equal to its period and effectively models this interference as $(T_S - C_S)$. This is a safe but pessimistic assumption. For the set of servers to be schedulable most if not all of the servers will have a response time that is shorter than their period. In particular, the highest priority server will typically have a response time equal to its capacity. (In considering this point it is important to distinguish between the response time of a Deferrable Server and the latest time it may execute due to the server's ability to suspend its execution if there are no tasks ready to execute).

2. The interference can be modeled as $(R_S - C_S)$. This removes much of the pessimism but does not provide exact analysis. The analysis given by Almeida in [8] can be made to match this model if an appropriate *"initial latency"* is used. Note this is not explicitly stated in [8].
3. The exact worst-case interference in the final server period is dependent on the amount of task execution that the server needs to complete before the end of the busy period. This may be much less that the server's capacity and so the maximum interference may be considerable less than $(R_S - C_S)$. The exact interference can be calculated using information about server priorities, capacities and replenishment periods.

Figure 3 illustrates the interference in the final server period.

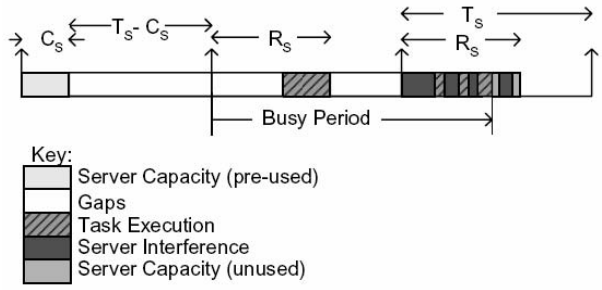

Figure 3 Interference in final server period

The extent to which the busy period *w* extends into the final server period is given by:

$$w - \left(\left\lceil \frac{L_i(w)}{C_S} \right\rceil - 1 \right) T_S \qquad (3)$$

Utilising the analysis of servers presented by Bernat and Burns [9] the interference due to higher priority servers in the above interval is given by:

$$I(w) = \sum_{\substack{\forall X \in hp(S) \\ servers}} \left\lceil \frac{w - \left(\left\lceil \frac{L_i(w)}{C_S} \right\rceil - 1 \right) T_S + J_X}{T_X} \right\rceil C_X \qquad (4)$$

Where *hp(S)* is the set of servers with higher priority than server *S* and J_X is the release jitter of the higher priority server *X*. (For a Deferrable Server, $J_X = T_X - C_X$, for a Periodic or Sporadic Server, $J_X = 0$).

Hence the full extent of the busy period is given by:

$$w = L_i(w) + \left(\left\lceil \frac{L_i(w)}{C_S} \right\rceil - 1 \right) (T_S - C_S) + \sum_{\substack{\forall X \in hp(S) \\ servers}} \left\lceil \frac{w - \left(\left\lceil \frac{L_i(w)}{C_S} \right\rceil - 1 \right) T_S + J_X}{T_X} \right\rceil C_X \qquad (5)$$

Note that the length of the busy period *w* appears on both sides of equation (5). This type of equation can be solved via a recurrence relation provided that the RHS is a monotonically non-decreasing function of *w*. It is not immediately obvious that this is the case here. However assuming that the servers are themselves schedulable, we observe that the interference in the server's final period, given by the summation term, is constrained to be between 0 and $(T_S - C_S)$. The summation term itself is a monotonically non-decreasing function of $L_i(w)$ except at values of $L_i(w) = nC_S$. At exactly these values the 2nd term increases by $(T_S - C_S)$, thus the 2nd and 3rd terms taken together form a monotonically non-decreasing function of the task load $L_i(w)$. The task load is itself a monotonically non-decreasing function of *w*, hence the RHS of the equation is a monotonically non-decreasing function of *w* and solution via a recurrence relation is possible although not entirely straightforward.

To solve equation (5) we need to modify the summation term to ensure correct convergence as intermediate values of *w* and $L_i(w)$ are calculated. This is as a direct consequence of the fact that the 3rd term alone is not a monotonically non-decreasing function of *w*. The modification simply ensures that the extent to which the busy period extends into the final server period is not considered to be an interval of negative length.

$$w_i^n = L_i(w_i^n) + \left(\left\lceil \frac{L_i(w_i^n)}{C_S} \right\rceil - 1 \right)(T_S - C_S) + \sum_{\substack{\forall X \in hp(S) \\ servers}} \left\lceil \frac{\max\left(0, w_i^n - \left(\left\lceil \frac{L_i(w_i^n)}{C_S} \right\rceil - 1 \right) T_S \right) + J_X}{T_X} \right\rceil C_X \quad (6)$$

Recurrence starts with a value of $w_i^0 = C_i + \left(\left\lceil \frac{C_i}{C_S} \right\rceil - 1 \right)(T_S - C_S)$ and ends either when $w^{n+1} = w^n$ in which case $w^n + J_i$ gives the task's worst-case response time or when $w^{n+1} > D_i - J_i$ in which case the task is not schedulable. (Where J_i is the task's release jitter relative to the server. This is zero for a bound task and $(T_S - C_S)$ for an unbound task).

3.2. Example

Consider a system comprising two Deferrable Servers with parameters given in the table below.

Server	C_S	T_S	J_S	R_S
HP	2	5	3	2
LP	8	20	12	16

The two highest priority tasks in the application serviced by the lower priority server (LP) are characterised as follows.

Task	C_i	T_i	D_i
1	10	50	50
2	8	100	100

The table below compares the response times of these tasks using, (1) the exact analysis introduced in this paper, (2) approximate analysis modelling interference in the final server period as $(R_S - C_S)$ and (3) the analysis given by Saewong et al in [5] treating interference in the final server period as $(T_S - C_S)$. For the purposes of this comparison, the tasks are considered as unbound.

Task	C_i	T_i	D_i	Response Times R_i		
				(1) Exact	(2)	(3)
1	10	50	50	38	42	46
2	8	100	100	82	84	88

This example clearly illustrates the improvements in task schedulability that can be obtained by using exact schedulability analysis.

If the tasks can be bound to the release of their server, then the worst-case response times can be reduced by $(T_S - C_S)$ to 26 and 70 respectively. This illustrates the benefit of binding tasks to the release of their server.

4. Evaluation

This section investigates the relationship between server replenishment period and the minimum server utilisation needed to achieve a schedulable system. In particular we examined the effect on the minimum required server utilisation due to different:

1. levels of server context switch overheads.
2. schedulability analysis techniques: Exact and 'sufficient but not necessary'.
3. server algorithms: Periodic, Deferrable and Sporadic servers.
4. tasks: bound and unbound to the server's period.

During the course of our investigation, we examined numerous synthetic applications. In this paper, two simple but representative examples are used for illustration purposes. Each of the examples comprises just two servers, one with a high priority, labelled HP and one with a lower priority, labelled LP.

We use a single higher priority server to represent additional load on the system. This means that the server of interest cannot simply use 100% of the processing time. Although only one higher priority server is used in our examples, the results are similar when multiple higher priority servers are present and constrain the available processor time within the replenishment period of the server of interest.

Example System #1. In our first example system, the higher priority server (HP) has a fixed capacity of 4 and a period of 10 time units (40% utilisation). The lower priority server (LP) is responsible for executing the tasks listed in Table 1 below.

Table 1

Priority	Exec. Time	Period	Deadline
1	5	50	50
2	7	125	125
3	6	300	300

Example System #1. In our second example system, the higher priority server (HP) has a fixed capacity of 10 and a period of 32 time units (31.25% utilisation).

Table 2

Priority	Exec. Time	Period	Deadline
1	8	160	100
2	12	240	200
3	16	320	300
4	24	480	400

The lower priority server (LP) is responsible for

executing the tasks listed in Table 2 above.

Our experimental investigation used a computer program to iterate over a range of possible periods for the lower priority server. For each server period, a binary search was used to determine the minimum server capacity commensurate with a schedulable system. For each server capacity examined, server schedulability was determined using existing analysis [9] whilst task schedulability was determined using the analysis derived in section 3.1. The results were plotted as graphs of minimum server utilisation against server period.

The graphs illustrate the importance of using exact analysis, choosing the most appropriate server algorithm, binding tasks to the server's period whenever possible and keeping server context switch overheads to a minimum.

4.1. Effect of overheads

There are two reasons why it is important to consider the effects of overheads when examining the choice of server periods and capacities. Firstly, the server implementation in any real system is likely to incur significant overheads. Secondly, from a theoretical standpoint, ignoring overheads leads to the conclusion that the optimal selection of server parameters involves selecting infinitesimally small values for the servers' periods and capacities.

The effects of server context switch overheads can be modelled by considering the server's capacity to be consumed first by context switch overheads and then by task execution. This is a safe if potentially slightly pessimistic approach to modelling overheads.

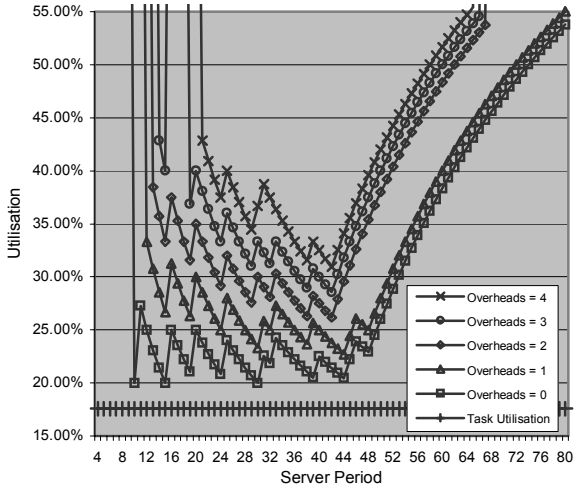

Figure 4 Overheads

Figure 4 illustrates the effect of server context switch overheads on example system #1, assuming that both HP and LP servers use the Deferrable Server algorithm. Figure 4 plots the minimum utilisation of the LP server necessary to achieve a schedulable system against the server's period for a range of server context switch overheads (0 to 4 time units). The total utilisation of the application tasks is 17.6% represented by the horizontal line immediately below the jagged lines depicting server utilisation.

From the graph, it is clear that overheads markedly increase the required server utilisation at short server periods. Hence, whilst the optimum server period is 10, 15 or 30 without taking overheads into account, it is 42 or 44 when the effects of overheads are included.

As server utilisation is simply C_S / T_S, server utilisation decreases with increasing period, until an increase in server capacity is required at which point server utilisation increases sharply, giving the characteristic saw-tooth shape.

It is interesting to note that the system remains schedulable for server replenishment periods that exceed the deadline of the highest priority task. This is somewhat counter-intuitive, however it is nevertheless correct. The server's relatively large capacity and short response time mean that it can schedule a task that has a shorter period than that of the server itself.

Once the server's period exceeds that of the highest priority task, each increase in server period requires a corresponding increase in server capacity to keep the interval from task release to the start of the task being serviced constant and hence the task schedulable. As the server period increases so its capacity is forced to increase causing the server utilisation to tend towards 100%.

At the far right hand side of the graph, as LP server utilisation begins to approach 60%, the LP server becomes unschedulable. (Recall that the HP server utilisation is 40%).

Similar graphs to Figure 4 have been produced for systems of more than two servers. The minimum feasible server period is limited by server schedulability: the server must be able to provide at least some capacity within its period. This is not possible for server periods that are less than the response time of the server with the next higher priority. The maximum feasible server period is also typically limited by server schedulability: for large replenishment periods, the server's capacity increases to the point where the server is again unschedulable.

4.2. Comparison of analysis methods

Figure 5 illustrates the minimum utilisation of the low priority server that was deemed necessary to schedule the task set from Table 1 using (1) the exact analysis presented in this paper and (2) the analysis of Saewong et al [5] which models interference in the

final server period as $(T_S - C_S)$. In both cases, the server context switch overheads were assumed to be 2 time units, hence the line on the graph for exact analysis is the same as the 'overheads = 2' line in Figure 4.

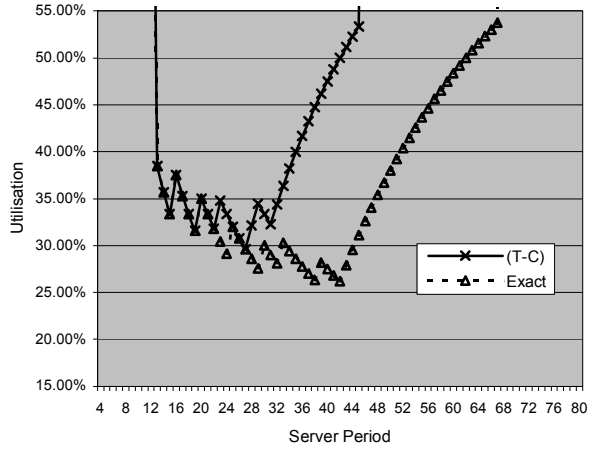

Figure 5 Comparison of analysis methods

Figure 5 provides a clear example of the difference that more precise analysis can make to the feasibility of a system. Using the exact analysis presented in this paper means that a server can be used which effectively requires 3.44% less processor utilisation, equivalent to 19.5% of the actual task load. Further the ability to use a longer server period reduces the time wasted due to server context switch overheads from 7.41% down to 4.76%.

4.3. Choice of server algorithm

The critical instant described in section 3.1 and the exact schedulability analysis given in this paper is applicable to Periodic, Sporadic and Deferrable Servers.

For all three server algorithms, the critical instant occurs when the server's capacity is exhausted as early as possible in its period, then there is a delay of $(T_S - C_S)$ before the server's capacity is replenished with subsequent capacity replenishments taking place after a period of T_S.

The only differences in analysis are as follows:

- When calculating interference from higher priority servers, Periodic Servers and Sporadic Servers have a jitter of zero whilst Deferrable Servers are treated as having a jitter equal to $(T_S - C_S)$ [9].
- Tasks cannot be bound to a Sporadic Server due to its non-periodic behaviour in anything other than the worst-case scenario.

Inspection of the exact analysis (equation (6)) shows that Periodic Servers *dominate* Deferrable Servers. That is there are no systems comprising a set of hard real-time application task sets that can be scheduled using a set of Deferrable Servers that cannot also be scheduled using an equivalent set of Periodic Servers with the same periods and capacities. There are however many sets of applications that can be scheduled using Periodic Servers that cannot be scheduled using Deferrable Servers. This is because the Deferrable Server has a drawback compared to a Periodic or Sporadic Server when used to service hard real-time tasks; the effect of *back-to-back* hits referred to earlier. Using a set of Deferrable Servers results in the lower priority servers receiving back-to-back interference from those of higher priority, increasing their response times and hence degrading the systems ability to schedule hard real-time applications.

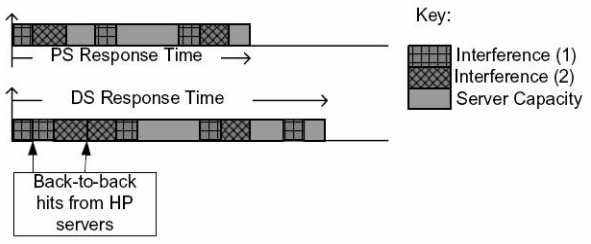

Figure 6 Response times due to Deferrable Servers

Figure 6 illustrates the longer worst-case response times of a system of Deferrable Servers due to back-to-back hits.

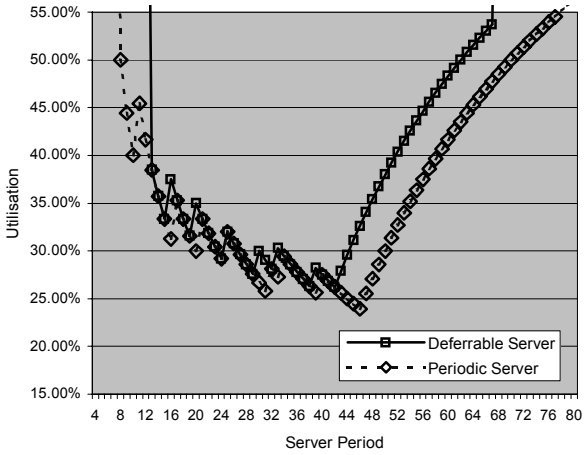

Figure 7 Comparison of server algorithms

Figure 7 compares the utilisation of server LP, required to schedule the task set from example system #1, when both HP and LP servers are (1) Deferrable and (2) Periodic.

As expected, the Periodic Server approach dominates the Deferrable Server algorithm. For short server periods (of 8-12 time units) and long server periods (of 68-77 time units), using Periodic Servers results in a schedulable system whereas using Deferrable Servers does not. This is a direct consequence of the back-to-back hit phenomenon. The minimum required server utilisation is 23.91% for the Periodic Server approach (period = 46, capacity = 11) compared to 26.19% for the Deferrable Server approach (period = 42, capacity = 11).

The same critical instant and exact schedulability analysis applies to systems comprising Sporadic Servers as it does to systems of Periodic Servers, with one key difference: tasks cannot be bound to Sporadic Servers and must therefore always be treated as unbound.

This means that Periodic Servers *dominate* Sporadic Servers. That is there are no systems comprising a set of hard real-time application task sets that can be scheduled using a set of Sporadic Servers that cannot also be scheduled using an equivalent set of Periodic Servers with the same periods and capacities.

Further, the Sporadic Server is far more complex to implement than the Periodic Server and so in practice the performance of a system based on Sporadic Servers would be inferior to that of a Periodic Server based system due to increased overheads.

We note that our analysis of Periodic Servers assumes that the Periodic Servers can service tasks that arrive after the start of the server's period. Effectively server capacity of at least $(C_S - t)$ is assumed to remain at time $t \le C_S$ from the start of the server period. This is a sensible model for many hierarchical systems, as each of the applications running on the system will typically contain an idle task that executes at a background priority level when all the application's other tasks are inactive. This idle task is often used to implement built-in-tests of the application and its memory areas and some types of watchdog functionality.

An alternative behaviour for a Periodic Server is for the server's capacity to be discarded at the start of its period if no tasks are ready to use it. We refer to these servers as *Discarding Periodic Servers*. Discarding capacity in this way reduces the server's ability to guarantee hard real-time applications. With this server behaviour, a critical instant occurs when at the start of the server's period its capacity is discarded and then the task of interest is released along with all other tasks of higher priority in the application. Effectively the jitter on unbound tasks is increased to T_S when a Discarding Periodic Server is used.

Both Deferrable and Sporadic Server algorithms were designed to provide responsive scheduling for soft aperiodic tasks in single application systems, whilst in the worst-case appearing to be similar to a periodic hard real-time task in terms of their effects on system schedulability. It is perhaps therefore not surprising that these mechanisms are no better than the much simpler Periodic Server approach when it comes to dividing up processor capacity between a number of hard real-time applications. It is a very different problem from the one for which they were designed.

Although we can recommend the use of Periodic Servers when the sole criteria is guaranteeing the deadlines of hard real-time application tasks, this does not mean that there is no place for Deferrable or Sporadic Servers in hierarchical systems. When quality of service (QoS) is also an issue, it may be appropriate to use a different approach.

4.4. Binding tasks to the server

Binding tasks to their server can improve system schedulability, effectively reducing the server utilisation required to schedule the task set. To illustrate the effect of making tasks 'bound' rather than 'unbound' we use example system #2. This system has task periods and deadlines chosen to emphasize the effect of having tasks bound to the release of the server. The task periods were chosen such they would be harmonics of a number of different server periods.

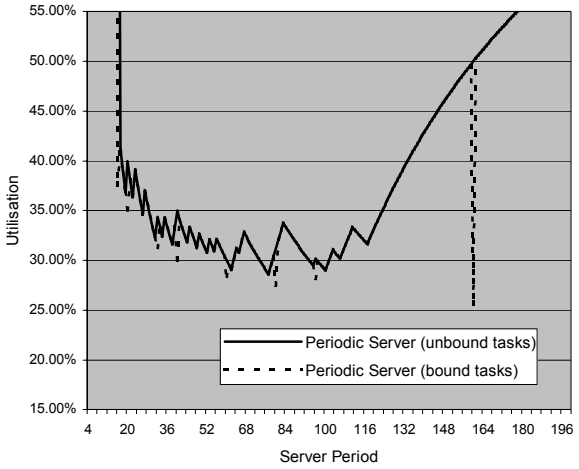

Figure 8 Bound and unbound tasks

Figure 8 shows the different server utilisations required to schedule the task set for a range of server periods. The two lines on the graph are both for Periodic Servers, however the dashed line illustrates the effect of binding tasks to the server whenever a task's period is an exact multiple of that of the server. This results in improvements in task response times and hence a system which is schedulable for lower server capacities. This is apparent from the graph for

server periods of 16, 20, 32, 40, 60, 80, 96 and 160.

Treating all tasks as unbound results in a minimum server utilisation of 28.57% (server period 77). Permitting tasks to be bound to the server reduces this minimum utilisation to 25.63% (server period 160).

We note from the shape of the graph that the problem of selecting the optimal server period does not lend itself to being easily solved via generic search techniques. The optimal server period, 160 in this case, is a single excellent solution surrounded by neighbouring solutions that are poor.

5. Summary and conclusions

In this paper we addressed the problem of scheduling a number of applications on a single processor using a set of servers. The motivation for this work comes from the automotive, avionics and other industries where the advent of high performance microprocessors is now making it both possible and cost effective to implement multiple applications on a single platform.

Our research has focussed on systems that are scheduled using fixed priority pre-emptive scheduling at both local and global scheduling levels.

5.1. Contribution

The major contributions of this work are:

- Exact response time analysis for hard real-time tasks scheduled under Periodic, Sporadic and Deferrable Servers. This analysis provides a reduction in the calculated worst-case response times of tasks compared to previously published work. A similar improvement is also apparent in the server capacity and replenishment periods deemed necessary to schedule a given task set.
- Extension of the analysis to tasks that are bound to the release of their server. We showed that permitting tasks to be bound to a server with the appropriate replenishment period always enhances task schedulability and can reduce the server capacity required.
- Comparison of Periodic, Sporadic and Deferrable Servers in terms of their ability to guarantee the deadlines of hard real-time tasks. The Periodic Server was shown to completely dominate the other server algorithms on this metric.

5.2. Future work

Today it is possible using the analysis techniques described in this paper to determine the optimal set of server parameters via an exhaustive search of possible periods and priorities for simple systems comprising 3 or 4 applications. Further work is required to provide an effective algorithm capable of choosing an optimal or close to optimal set of server parameters given systems comprising perhaps ten or more applications.

Another interesting area of future research involves incorporating Quality of Service (QoS) requirements into hierarchical fixed priority pre-emptive systems. Finally extension of this work to multiprocessor platforms requires careful consideration.

6. Acknowledgements

This work was partially funded by the EU Information Society Technologies (IST) Program, Flexible Integrated Real-Time Systems Technology (FIRST) Project, IST-2001-34140 and the UK EPSRC funded DIRC project.

7. References

[1] T-W. Kuo, C-H. Li. "A Fixed Priority Driven Open Environment for Real-Time Applications". *In proceedings of the IEEE Real-Time Systems Symposium*. Madrid, Spain, December 1998.

[2] Z. Deng, J.W-S. Liu. "Scheduling Real-Time Applications in an Open Environment". *In proceedings of the IEEE Real-Time Systems Symposium*. December 1997.

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

[4] C.L.Liu, J.W.Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment" *JACM*, Vol. 20, No. 1 January 1973, pp. 46-61.

[5] S. Saewong, R. Rajkumar, J. Lehoczky, M. Klein. "Analysis of Hierarchical Fixed priority Scheduling". *Proceedings of the ECRTS*, pp. 173-181, 2002.

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

[7] G. Lipari, E. Bini. "Resource Partitioning among Real-Time Applications". *Proceedings of the ECRTS*, Portugal, July 2003.

[8] L. Almeida. "Response Time Analysis and Server Design for Hierarchical Scheduling". *Proceedings Real-Time Systems Symposium Work-in-Progress 2003.*

[9] G. Bernat, A. Burns. "New Results on Fixed Priority Aperiodic Servers". *Proceedings of the IEEE Real-Time Systems Symposium*. Phoenix, Arizona, December 1999.

[10] N.C. Audsley, A. Burns, M. Richardson, A.J.Wellings. "Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling". *Software Engineering Journal*, 8(5) pp. 284-292, 1993.

[11] EU Information Society Technologies (IST) Program, Flexible Integrated Real-Time Systems Technology (FIRST) Project, IST-2001-34140.

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

[13] R.I.Davis, A. Burns "Hierarchical Fixed Priority Pre-emptive Scheduling" *Technical Report YCS-385* Department of Computer Science, University of York, April 2005.