New Results on Fixed Priority Aperiodic Servers

Guillem Bernat and Alan Burns
University of York
Real-Time Systems Research Group
University of York
YO1 5DD - York (UK)
[bernat,burns]@cs.york.ac.uk

Abstract

The issue of using the Sporadic Server (SS) for scheduling aperiodic tasks has received new attention under the POSIX standard as it has been proposed in P1003.1d, the additional real-time extensions to POSIX. The SS has been traditionally considered a better approach to the deferrable server (DS) due to its supposed higher achievable utilisation. However, SS also has higher implementation complexity. Nevertheless, the analysis of the comparisons performed from several authors between DS and SS is not conclusive. A review on fixed priority servers is presented with a new parameter selection technique and comprehensive performance analysis based on simulation techniques. With this parameter selection, it is shown that no server performs significantly better than the other in most of the situations. This suggests that future POSIX revisions for real-time support should also consider mechanisms by which other types of servers could be implemented.

1. Introduction

A real-time system is usually constructed as a set of concurrent tasks which may have hard or soft deadlines. Hard tasks have strict timing constraints, soft tasks only require good average response times (here the term aperiodic task is also used as a synonym of soft task). A real-time system with this mixture of hard and aperiodic tasks has to satisfy that:

- All hard tasks have to meet their deadlines, even in the worst-case. This requires schedulability tests to be provided that account for the interference that a task can suffer from the rest of the tasks in the system.
- Aperiodic tasks have to have a *good* response time. Here, *good* should mean that the task meets a soft deadline in most of the invocations. However, to the best of our knowledge, analytic solutions for determining whether arbitrary mixes of aperiodic tasks with hard tasks will meet their response time requirements do not exist [24, 19]. The traditionally used objective is to consider a single aperiodic task source only, and to minimise its average response time.

The principal problem of dealing with aperiodic tasks is that the worst case execution time may not be known (or be extremely large compared with its average computation time) or it may not be possible to determine lower bounds on the interarrival time of aperiodic requests. If the worst case computation time is known and there is a minimum interarrival time, then aperiodic tasks could be implemented and analysed as hard tasks. If we can not make these assumptions then different approaches have to be provided to prevent the aperiodic tasks from making the hard ones missing their deadlines.

The easiest (and less efficient) way to prevent soft task from interfering in an unpredictable way with hard tasks is to schedule them as background tasks. Several approaches have been devised to improve the responsiveness of the soft tasks which are revised in the following section. The ones of particular interest for us are *aperiodic servers*. The idea of the server approach is to have a special task, usually at the highest priority, that performs the soft operations. To bound the amount of interference it may produce on low priority hard tasks, the server has a maximum *capacity* (also called budget or time tickets) that can be used to schedule the aperiodic task. If there is aperiodic work pending and there are available tickets, the aperiodic task executes until it finishes or no more time tickets are available. The server also requires a mechanism to refill the capacity and therefore when the server will be entitled to run again. The difference between these aperiodic server approaches lies on the way the capacity of the server is replenished and by how much, which also determines the maximum capacity the server may have.

Several server models have been introduced for fixed priority systems. The simplest one (in implementation complexity and run-time overhead) is the deferrable server. Other approaches like the priority exchange server or the sporadic server although they require a more complex implementation they were designed to achieve better performance. Previous work leads to the conclusion (which is discussed in this paper) that the sporadic server was the best server because it had a better performance than the deferrable server and a similar performance to the priority exchange server but with lower implementation and run-time overhead.

The issue of comparing the performance of different servers has been brought to attention under the POSIX (Portable Operating System Interface) standard P1003.1 [10]. POSIX provides a standard API to the operating system, of particular importance to the real-time community are the real-time related POSIX standards: P1003.1b Real-time and I/O extensions [11] (was: POSIX.4) (this document has been amended by P1003.1i [14]); P1003.1c Threads Extensions [12] (was: POSIX.4a), and P1003.1d additional real-time extensions (which is still a draft)) [13] (was: POSIX.4b).

In its current form, the P1003.1d standard provides support for the sporadic server only, and implementations of the other servers (for instance the deferrable server one) have been discarded. As we show that both servers have a similar performance (even without considering run-time overheads, which are larger for the sporadic server than the deferrable server), we would suggest that the deferrable server should be supported in a future extension of the POSIX standard. Or, more generally, a generic mechanism by which different servers could be implemented.

1.1. Related work

We now present the main approaches that deal with the problem of scheduling a mixture of aperiodic tasks and hard tasks and we postpone the details on the operation of the servers for section 3.

The Deferrable Server (DS) and Priority Exchange Server (PE) were first introduced in [17] under a fixed priority scheduling framework to solve the poor performance of background and polling approaches. The extended version of the priority exchange server was later described in [20]. A new type of server, the Sporadic Server (SS), was introduced in [21]. These papers presented schedulability tests based on utilisation bounds on the server. The main conclusion of their work was that the sporadic server was superior to other servers because the maximum utilisation achievable by the SS was larger than the DS one. They also showed that the PE approach has no advantage over the SS approach but requires a more complex implementation, therefore, in the rest of the paper, the discussion is centered only around the DS and the SS approaches.

Simulation studies showed that these approaches effectively achieved low average response times of the aperiodic tasks. Later, an exact analysis of the deferrable server was presented in [24] with more accurate utilisation bounds on the server. An analysis of the conclusions from these papers form an important part of our contribution and are reviewed in detail later in the paper.

The issue of using aperiodic servers under dynamic priorities has also received attention. Ghazalie *et al.* in [9] introduced dynamic priority versions of the deferrable server, priority exchange server and sporadic server. At the same time, Spuri *et al.* in [22] also introduced five different servers under dynamic priorities. The issue of using dynamic servers and its variants has received continuous attention, for example see [23, 6]. The work of Ghazalie *et al.* in [9] also considers exact schedulability tests for fixed priority servers in order to compare them with their dynamic versions. Also in that paper there is the first reference to the problem of *double blocking* which is reviewed and discussed in detail in section 4.1.

A different approach for scheduling aperiodic tasks was the one based on the concept of "slack stealing" introduced in [16] and extended later by [18] to include guarantees on hard tasks. Their results were corrected and generalised in [7]. Finally, another mechanism for addressing the problem was by means of the dual priority mechanism, first introduced in [5, 8] and later extended in [3].

In this paper we concentrate only on fixed aperiodic servers because it is the only service being standardised in POSIX. In particular we only compare the DS and SS approaches. Similar analysis may be undertaken for the PE server.

We believe that the previous studies that compared both servers are inconclusive. For this purpose we investigated in detail current schedulability tests, server configuration parameters and performance studies. This paper is the result of the review of such studies. Based on the weakness we found, we have proposed a new exact schedulability tests that includes the additional interference from the double blocking of servers, extensive simulation studies and more important, a new parameter selection technique that improves significantly the performance of both servers. This new parameter selection results in both servers having a similar performance.

For this purpose we have structured the paper in the following way. First we introduce a process model used in the rest of the paper. We then review the work previously done on aperiodic servers. After that section we present our analysis that include new scheduling tests, a study of the maximum server utilisation, new performance analysis studies and finally a parameter selection procedure. Finally, using the results on our analysis we review previous misconceptions on aperiodic servers.

2. Process model

We consider the traditional process model in which a system is made up of n periodic and sporadic hard tasks, $\Pi = \{\tau_1, \ldots, \tau_n\}$ and an additional task, the server task τ_s. Each task τ_i is a sequence of requests for execution of a particular computation which are characterised by a tuple $\tau_i = (C_i, D_i, T_i, P_i, B_i, C_{i,b}, J_i)$, where C_i is the worst-case computation time of the request; T_i is a lower bound on the minimum interarrival time (this is called the period of the task for both periodic tasks or sporadic tasks); D_i is the deadline of the task relative to the request. If the task τ_i is invoked at time t then it should have finished by $t + D_i$. The earlier models assumed $D_i = T_i$, we later show schedulability tests for $D_i \leq T_i$; P_i is the priority of the task, 1 being the highest. We also assume that no two tasks share the same priority; B_i is the blocking factor. It is the maximum interference a task may suffer from lower priority tasks due to the operation of a protocol for shared resources or access to a critical section. We also denote by $C_{i,b}$ the worst-case computation time task i spends in resource b (i.e. critical section or protected object); J_i is the release jitter. It is the maximum elapsed time between the event that awakes the task occurs and the time instant at which the task is in fact ready to run. Release jitter is usually zero for periodic tasks.

The server task requires the following additional parameters: C_s, the maximum capacity of the server; T_s, the replenishment period. When distinction between the DS and SS is required it will be denoted with a superindex *i.e.* C_s^{DS}.

The utilisation of the periodic tasks, U_p is

$$U_p = \sum_{\tau_i \in \Pi} \frac{C_i}{T_i}$$

In the same way, the utilisation of the server is given by $U_s = \frac{C_s}{T_s}$. We first assume that there is a single server at the highest priority. We later relax this hypothesis allowing a server at any priority level.

The servers are used to process aperiodic tasks. The computation requirements of these aperiodic tasks are usually described in terms of a poisson process, described by two parameters. The average computation time, denoted by $\bar{C}_s$ and the average interarrival time, denoted by $\bar{T}_s$. The average requested load by the aperiodic is then $\bar{U}_s = \frac{\bar{C}_s}{\bar{T}_s}$.

3. Aperiodic servers: A review

We now introduce previously published results on schedulability tests of aperiodic servers. We leave the discussion of such results for the following section.

A task scheduled with a DS has its capacity replenished to the maximum value of C_s^{DS} at fixed time instants $t = kT_s^{\mathsf{DS}}$ for $k \geq 0$. Whenever an aperiodic task arrives and there is capacity available, it starts executing until it finishes or the capacity is exhausted. If it does not consume all the capacity, the remaining time is available for the next aperiodic task. If it runs out of capacity the task is suspended until the capacity of the server is replenished at the next server replenishment.

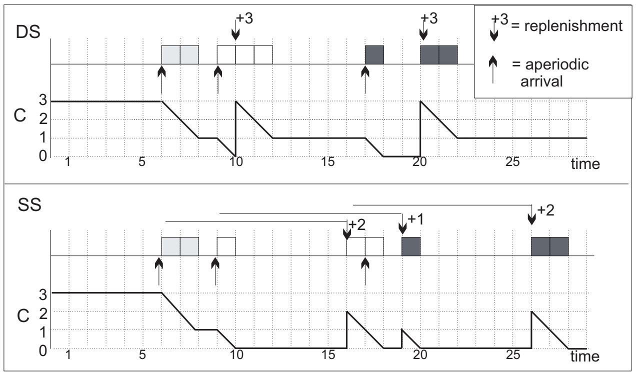

Figure 1. Operation of servers for three aperiodic arrivals.

The DS approach may block lower priority tasks as much as $2C_s^{\mathsf{DS}}$ consecutive time units. This is because the capacity of the server is replenished at a fixed time instant, while an aperiodic request may occur at any instant within the period. If the aperiodic task arrives just C_s^{DS} time units before the replenishment instant then it will use C_s^{DS} time units plus another C_s^{DS} time units from the next period. This effect is called *the double hit*. Note that further replenishments are spaced by T_s^{DS} time units from the first one.

The operation of the SS differs from DS by when the replenishment is performed and by how much. The operation of the server is as follows. The capacity of the server is replenished exactly T_s^{SS} time units after the arrival of any aperiodic request. If the aperiodic task consumes c time units then the server will have its capacity increased in c time units exactly T_s^{SS} units after the aperiodic request arrived. Note that if the task requires more computation than the maximum capacity available then it will execute until the capacity is exhausted, and it will be suspended until a replenishment occurs. [1]

The operation of both servers can be seen graphically in figure 1. The graph displays the operation of both servers under three arrivals of an aperiodic task at time instants $t = \{6, 9, 17\}$ which require $2, 3$ and 3 time units of computation time. The top graph shows the operation of the DS, the bottom part the SS one. The graph also displays the evolution of the available capacity. We assume that both servers have the same capacity $C_s = 3$ and replenishment

[1] The original SS algorithm presented in [21] actually is more subtle and allows the schedule to replenish earlier if the aperiodic task arrives while a higher priority task is running. However Michael Gonzalez-Harbour in a private communication noted us that this could result in lower priority tasks missing their deadlines.

period $T_s = 10$.

3.1. Schedulability analysis

We first turn to the problem of determining whether hard tasks meet their deadlines when scheduled together with aperiodic servers. Two additional problems are derived from this first one: the double blocking problem that occurs when aperiodic tasks may use shared resources, and parameter selection.

3.1.1 Utilisation based schedulability tests

Lehoczky *et.al.* in [17] show that a lower bound on the maximum utilisation of periodic tasks U_p in terms of the maximum utilization of the server U_s for the DS and SS approaches is given by:

$$U_p^{\mathsf{SS}} = \ln \frac{2}{U_s^{\mathsf{SS}} + 1} \qquad U_p^{\mathsf{DS}} = \ln \frac{U_s^{\mathsf{DS}} + 2}{2U_s^{\mathsf{DS}} + 1} \quad (1)$$

This can be rewritten with U_s as a function of the periodic utilisation U_p:

$$U_s^{\mathsf{SS}} = \frac{2}{e^{U_p^{\mathsf{SS}}} - 1} \qquad U_s^{\mathsf{DS}} = \frac{2 - e^{U_p^{\mathsf{DS}}}}{2e^{U_p^{\mathsf{DS}}} - 1} \quad (2)$$

The graph of this functions is plotted later in the paper in figure 2.

In general, for a given periodic load U_p the maximum utilization of the server is larger for the SS than for the DS. The difference between server sizes is only significant for low to medium periodic loads, a difference of more than 6% in server sizes occurs only when periodic loads are between 22% and 54%, with a maximum difference of 10%. This utilisation bounds lead to sufficient but not necessary schedulability tests. There are task sets that are schedulable with a server with larger aperiodic utilisation than the values obtained by the previous equations (in fact there is a large difference between this bounds and values found by exact tests as section 4.2 shows).

One of the most interesting conclusion of the SS server is that it has been proven that it can be modeled as a periodic task [21]. Therefore any schedulability test used for periodic tasks could be also used for determining the schedulability of a task set with sporadic servers.

3.1.2 Response time based tests

The main drawback of utilisation based tests is that they summarise the information of a task set into a single number, the utilisation, which is then used to determine the schedulability of the system. This approach does not exploit the huge variability between the different task sets. Simple exact tests based on worst-case response time analysis of each task set do exist for both aperiodic servers and are, therefore, preferred.

The worst case response time formulation for the DS and SS was presented in [9]. It is based on computing the maximum interference a server can produce over lower priority tasks in interval $[0, w]$ (assuming $J_i = 0$ for all tasks) [2]:

The maximum interference a DS server can produce over lower priority tasks was shown to be:

$$\left(1 + \left\lceil \frac{w - C_s}{T_s} \right\rceil\right) C_s \quad (3)$$

Traditional worst case response time formulation can be used to compute the worst case response time for a task τ_i [4]. It is given by, $R_i = w$ where $w > 0$ is the smallest solution to the following equation:

$$w = C_i + \sum_{j \in \mathsf{hp}(i)} \left\lceil \frac{w}{T_j} \right\rceil C_j + \left(1 + \left\lceil \frac{w - C_s}{T_s} \right\rceil\right) C_s \quad (4)$$

where $\mathsf{hp}(i)$ is the set of tasks of higher priority than task τ_i. This equation can be solved with a recurrence approach, starting with $w^0 = C_i$ and finishing when $w^{n+1} = w^n$. This equation has a solution if $U_p + U_s \leq 1$.

For the SS, as there is not the problem of double hit, the interference produced by the server is simply :

$$\left\lceil \frac{w}{T_s} \right\rceil C_s \quad (5)$$

This exact test can then be used to determine whether a given task set with a server with parameters C_s, T_s is schedulable. If $R_i \leq D_i$ for all tasks then all tasks meet their deadlines and the task set is schedulable.

3.2. Server parameter selection

The analysis and simulation studies performed by the previously cited authors show that the selection of the parameters of the server have a big impact on the average response time of the aperiodic load. To achieve low average response times of the aperiodic task it is desired:

- To have large server utilisation.
- To have large values of C_s. This is due to the fact that the probability of an aperiodic request exhausting the capacity of the server decreases as the capacity becomes larger.

[2]The same analysis appears in [24]. However, even though their equation number 2 is, in essence, correct, there are some minor flaws in the schedulability tests of their equations 3 and 4 and in successive references: the expression $\left\lceil \frac{t - T_0}{T_0} \right\rceil$ should be changed by $\left\lceil \frac{t - C_0}{T_0} \right\rceil$.

- To run the server at a high priority(usually the highest)[3]

The method they use for selecting the parameters for the servers (DS and SS) is to first select a period (usually the largest so that the task set is still in rate monotonic order, *i.e.* $T_s = T_1$) and then find the maximum capacity C_s that satisfies the schedulability of the rest of the tasks in the system using some schedulability test. This can be done with a bisection search on the minimum and maximum values.

Strosnider *et al.* in [24] identify that it may be the case that a server with a period shorter than T_1 may have a higher utilisation due to the favourable phasing of the server with the rest of the tasks in the system. They also raise the issue of not using the server at all when average requested computation time is larger than the budget because under this situation the polling server may be more adequate because it may have a larger server size.

As it will be shown later in section 4.4, the selection of $T_s = T_1$ is not adequate. In fact, values much larger than T_1 can be generally used which result in larger server capacities and in better performance. These values still guarantee the schedulability of the rest of the system even though task sets are not in deadline monotonic order.

3.3. Performance Analysis

Previous work on DS and SS have performed simulation studies to compare the relative performance of the different servers [9, 17, 21, 24]. Their simulation studies analysed the average response time of the different approaches for different periodic and aperiodic loads. The simulations were performed assuming a correct but restricted model. Their simulation are characterised by:

- Few task sets were used for the simulation (or even just a single one). The variability of the operation of the servers for different task sets was not mentioned.
- Server periods were fixed to be equal to the shortest period of the rest of the tasks, and C_s computed as the maximum value that satisfies the schedulability. No analysis on the implication on using different server capacities was reported.
- Short aperiodic computation times with relation to server size. No study addressed the relative performance of DS and SS when aperiodic computation times were higher and possibly even larger than the available capacity, which is likely to occur when high priority periodic tasks have small periods (therefore limiting the maximum size of the server). Also we can not make assumptions on the source of the aperiodic load which can have very distinct computation times.

The conclusion derived from the previous analysis was that SS was better than DS. This was based on:

- SS can be treated as a periodic task (the DS not).
- SS has higher utilisation than DS
- SS allows larger capacities

In section 4.3 we present complementary simulations that show the variability of the performance for different parameter configurations for different task sets (even with the same utilisation) and for larger ranges of aperiodic computation times.

We show in the following sections that these assertions are not accurate. We show that both DS and SS can be treated as periodic tasks. We also show response time schedulability tests that can be applied to both servers (even for multiple servers at any priority) which lead to an improved technique to assign parameters to the server that results in much higher utilisation of the server that previous models and which results in almost identical maximum utilisation for the DS and SS. These simulations ignore overheads, but SS will always be more inefficient to implement.

4. Schedulability analysis

We now present a reformulation of the schedulability tests for both servers, we describe the double blocking problem and new performance studies which allows us to provide a parameter selection technique.

The schedulability test can be rewritten in a more elegant way by noting that in the worst case a DS behaves like a periodic task with a jitter equal to $J_i = T_i - C_i$. The proof of this is straightforward. It has already been shown that the worst case response time for lower priority tasks occurs when at the critical instant, the aperiodic server is requested in a way that a double hit occurs and then it is requested as fast as possible. A double hit occurs when the server is invoked just C_i time units before a replenishment occurs. And the worst case interference over lower priority tasks occurs if this instant is also the release time of all lower priority tasks. This is, in fact, the same definition of the scenario of release Jitter (see, for example [1]) in which the server task is modeled as a periodic task with a Jitter equal to $J_i = T_i - C_i$ which will result with the double hit.

The worst-case response time of a task τ_i, is given by $R_i = J_i + w$, where w is the maximum value greater than zero that is a solution to the following equation:

[3] The issue of setting the server at lower priorities to benefit from larger capacities has not been investigated. We have found that for task sets with $T_i = D_i$ there is no gain on setting the server at low priorities as most of the available slack can be given to the server even when it is set at the highest priority. However, for task sets where D_i may be smaller than T_i, there may be very little slack at the highest priority and therefore the capacity of a server at the highest priority may be very small. The capacity may be much larger if the server runs at lower priorities. This issue needs further investigation.

$$w = C_i + \sum_{j \in \mathsf{hp}(i) \cup \{s\}} \left\lceil \frac{w + J_j}{T_j} \right\rceil C_j \qquad (6)$$

It is easy to show that equations (4) and (6) are equivalent if we consider $J_i = 0$ for all non server tasks. If we substitute J_j by $T_j - C_j$ in equation (6) we have:

$$\left\lceil \frac{w + J_j}{T_j} \right\rceil = \left\lceil \frac{w + T_j - C_j}{T_j} \right\rceil = 1 + \left\lceil \frac{w - C_j}{T_j} \right\rceil \qquad (7)$$

Therefore equation (6) can be used as an exact schedulability test for both the DS and SS servers considering them like any other task (for the SS just consider $J_s = 0$). Any scheduling model that applies to sporadic tasks with jitter can be also applied to sporadic servers.

With this schedulability tests a server can be configured with much higher utilisation than the bounds provided in equation (1). This is due to the fact that the particular task set characteristics are used to determine maximum capacities and periods instead of having to provide upper bounds for all possible task sets. This issue is investigated in the following section.

4.1. Double blocking problem

Using a protocol for accessing shared resources within an aperiodic server results in a problem we call the *double blocking problem*, first described in [9]. If the server budget is exhausted during a critical section then it may overrun by the amount of time spent in that critical section. This is because even though the capacity is exhausted, it inherits the priority of the protected object. A lower priority task τ, can then suffer two blockings, one due to the normal operation of the protocol (the B_i factor), and one due to the overrun (the time the server task overruns its capacity). This additional interference should be accounted for in the schedulability test.

To prevent this effect from carrying over to the next invocations, Ghazalie and Baker propose modifying the operation of the servers by reducing the amount of capacity replenished by the amount overrun in the previous period. With this simple rule a lower priority task can only suffer one overrun from a server task.

It is easy to update equation (6) to account for blocking factors, including double blocking. We denote by B'_s the amount of overrun a server s may have due to access to a critical section. The maximum overrun occurs just before the capacity is exhausted and when the server enters a critical section. Therefore the maximum overrun is in fact the length of the longest critical section:

$$B'_s = \max_{\forall b \in \mathsf{uses}(s)} \{C_{s,b}\}$$

where $\mathsf{uses}(s)$ is the set of critical sections used by aperiodic task s. B'_s is zero for servers that do not access critical sections or for non server tasks. The schedulability test of equation (6) has to be updated to account for the double blocking as well. It is easy to see that the worst interference a server task may produce on lower priority tasks occurs when the task enters the worst case critical section just before the capacity is exhausted. In this case the amount of computation used in the first invocation is $C_s + B'_s$, at the second replenishment the capacity is replenished by $C_s - B'_s$ only, however if the server enters the critical section, again, just before the capacity is exhausted then the maximum computation used is $C_s - B'_s + B'_s = C_s$. The same situation occurs for successive replenishments. Therefore, the worst case response time formulation becomes:

$$w = C_i + B_i + B'_s + \sum_{j \in \mathsf{hp}(i) \cup \{s\}} \left\lceil \frac{w + J_j}{T_j} \right\rceil C_j \qquad (8)$$

This equation does not hold if $B'_s \geq C_s$. For non server tasks $C_{i,s}$ is always smaller than C_i because it is in fact part of the computation. This is not necessarily the case for servers as C_s is the server capacity which can be arbitrarily small. If so, then multiple replenishments are required to bring the capacity to a positive value. This implies that in the worst case, the server behaves as if it was being replenished with a larger period. This can be easily accounted for in the formulation with simple period transformations.

4.2. Maximum server utilisation

Our first analysis of the exact schedulability test suggested that larger server utilisation than the utilisation based tests could be achieved by exploiting the particular characteristics of each task set. For this purpose a set of experiments were performed to determine the maximum server utilisation of the DS and SS using the exact schedulability test presented above.

The first problem is to find the pair of parameters T_s, C_s of the server that maximise the utilisation and that guarantee the schedulability of the rest of the tasks in the system. There is a maximum on the possible range of capacities C_s independently of the replenishment period chosen. We denote this maximum C^*. For each possible capacity $C_s \in [1, C^*]$, we can then find the minimum server period with a bisection search (and also with binary exponential backoff). We can then choose the pair C_s, T_s that maximises the utilisation.

The problem is then finding C^*. This value corresponds to the amount of time a task at the highest priority could run without making any low priority task miss its deadline. This is in fact, the smallest of the total amount of idle time at each priority level between $t = 0$ and the D_i. If we consider a task τ_i with deadline D_i at priority $P_i = p$, the idle time

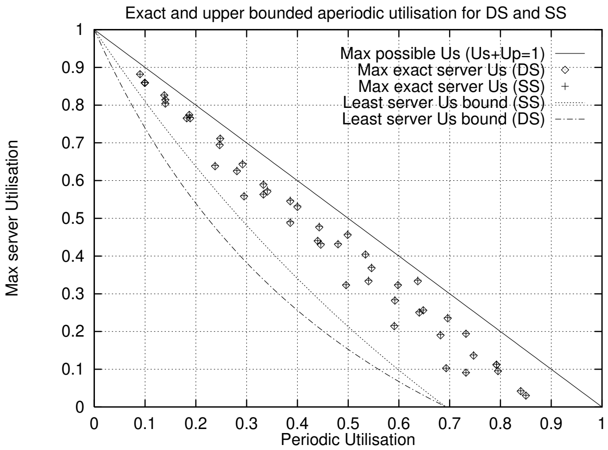

Figure 2. Maximum server capacities for DS/SS for different periodic utilisation.

at level p is the amount of time a task running just below τ_i could use so that it would finish before or at D_i. This is equivalent to the amount of time a higher priority task could run so that τ_i would finish by the deadline. It is easy to show that this is the worst case scenario and therefore if τ_i finishes by the deadline at the critical instant, it will always meet its deadline.

The idle time at priority p in $[0, t]$, denoted by $\delta_p(t)$, can be computed in several ways. One of the most effective is by introducing a virtual task τ' at priority just below p and considering $T' = D' = t$. The maximum value C' so that the task is schedulable is the amount of idle time at level p. This value can be easily computed with a binary search. The method can be applied to a wide range of process models and it can be optimised by using upper and lower bounds on the amount of idle time. For a complete analysis see [2].

Therefore, the maximum capacity is:

$$C^* = \min_{\forall \tau_i \in \Pi} \{\delta_{p_i}(D_i)\}$$

For the DS the maximum capacity is at most $C^*/2$ due to the double hit. This technique was applied to 200 task sets, made of 8 tasks each, that were randomly generated with utilisation in the range of 10% to 90% and with periods in the range $[100, 10000]$. The procedure to generate a task set of n tasks with utilisation U is as follows. First of all periods were randomly uniformly generated, then computation times were randomly generated following and exponential distribution with mean $\frac{(U-U_i)T_i}{n-i+1}$, where $i \in [1, n]$ is the index of the task, U_i is the utilisation of the tasks generated so far. Task sets that differed in more than 2% of the desired utilisation or that were not schedulable were discarded. After that, the task set was sorted in deadline monotonic order.

For each task set an additional server task was added and server parameters computed that lead to maximum utilisation. Figure 2 shows the maximum server utilisation of these task sets. The x axis is the periodic task set utilisation and the y axes is the maximum utilisation of the server. Each point in the graph corresponds to one of these task sets. A cross represents the maximum server utilisation for the SS a diamond is used for the DS. As the maximum server utilisation is almost the same for the DS and SS, each point in the graph appears to be a crossed diamond. The graph also shows the utilisation bounds of equations (1) rewritten as a function of the periodic load.

Two main conclusions are derived from this analysis:

- The maximum utilisation of both DS and SS are much higher than the utilisation bounds derived by the equations in (1). For example for a periodic utilisation of $U_p = 0.6$, the maximum utilisation bound is around 0.1 whereas the exact test lead to utilisation U_s between 0.2 and 0.3. This is not surprising because these equations determine the worst case for *any* possible task set while the exact analysis finds the value tailored to each particular task set.

- There is almost no difference in maximum server utilisation.

This implies that both DS and SS do have maximum utilisation which are almost the same, and which lead to a high total utilisation. The same properties were observed for different task set configurations with shorter and larger periods and with few or a large number of tasks, in fact, 5,000 task sets were generated and in all of the cases the difference between maximum utilisation of the DS and SS was below 1%.

The only difference between DS and SS, which is not shown in the graph, is that the maximum utilisation of the DS happens for small server capacities, whereas the SS may achieve the maximum at higher capacities. The implication that server capacity has on the performance of the server is discussed in detail in the next section.

The important conclusion of this result is that any comparison between DS and SS can not be based on differences in server utilisation as their maximum server utilisation are practically identical. The factor that greatly determines the performance of the servers is the capacity. For this purpose extensive simulations studies were performed which are summarised in the following section. This then lead to rules for determining the parameters of the servers that lead to best performance.

4.3. Performance Analysis

In order to test the performance of the different servers, extensive simulation experiments were performed under a wide range of configurations and for a large number of task sets. The experiments were defined to study:

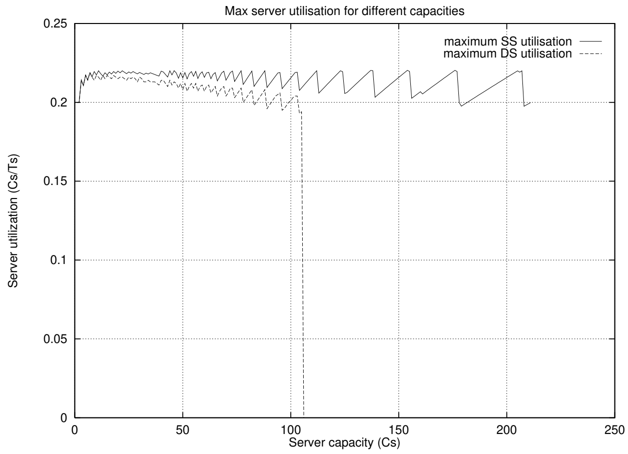

Figure 3. Maximum server utilisation for all possible server capacities, for a task set with $U_p = 0.69$.

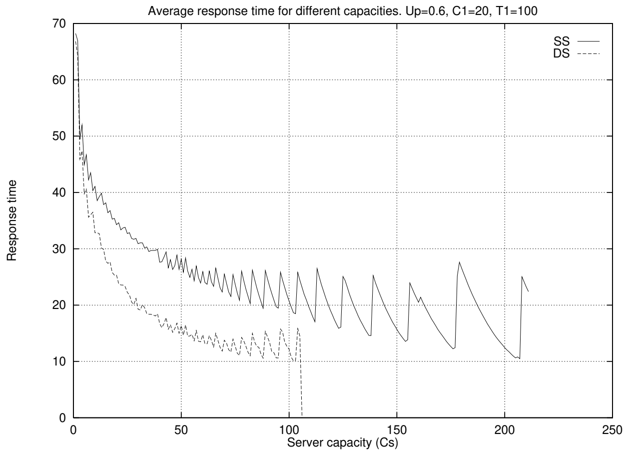

Figure 4. Average response time of 4. ($\bar{C} = 10$, $\bar{T} = 100$)

- the difference on server utilisation for different capacities,
- the impact on the average response time of the soft tasks, and
- the relative performance of both servers for multiple task sets.

The task sets were generated in different ways and with different number of tasks. However, all different methods lead to similar results. The task sets used in the graphs shown in this paper were used following the procedure described in section 4.2.

Each task set had a server as the highest priority task, either DS or SS. We did not allow a server task to run as background task when the capacity was exhausted, this described better the performance of the server alone. We assume that each server was providing service to an aperiodic task which was a poisson process. Interarrival times and computation times followed exponential distributions. Each simulation was carried out for 500,000 ticks. To be able to compare DS and SS, aperiodic arrival instants and computation requests were reused so that both servers had to handle the same aperiodic traffic.

4.3.1 Capacity

The first experiment was designed to determine the range of parameters that could be set for the servers, more specifically, the range of capacities and what was the effect of these capacities on the average response time of the server task[4].

[4]When we refer to the average response time of the server task we mean the average response time of the aperiodic task that uses the server.

We computed C^{DS*}, and C^{SS*} for a wide range of task sets, and for all possible capacities, c, we computed the minimum server period. We then simulated the task set under different load conditions. The results of one of these simulations are shown in figures 3 and 4. The task set corresponds to the 69% utilisation task set described in [9][5] with an aperiodic load of 10% by an aperiodic task with $\bar{C} = 10$ and $\bar{T} = 100$. The x axis is the capacity of the server, for figure 3 the y axis is the maximum utilisation computed with the procedure outlined in section 4.2. Figure 4 shows the average response time for each possible server capacity.

Some interesting results can be extracted from these two graphs:

- The utilisation attainable depends on the selected capacity and has a saw shape. This shape is due to the ceiling function in the schedulability test. This corresponds to the number of full invocations that fit in the first critical instant of the rest of the tasks.
- The server period can be much larger than the shortest period of any other task (even though the task set is not in deadline monotonic priority ordering). The schedulability test assures that the rest of the tasks are schedulable. This shows that server capacities can be much larger than they could be if we limited the server period to be equal to the shortest period of the other tasks.
- The maximum server utilisation for the SS server can be attained at different capacities. For the DS one, the maximum occurs at low capacities. This behaviour of the DS server is due to the impact of the initial double hit. The larger the capacity, the larger the double hit and therefore the lower the total utilisation.

[5]Each number is divided by 10.

- The response time greatly depends on the server capacity. The DS achieves lower response times than the SS does for a given capacity. This, again, is due to the double hit. As the DS is able to serve the second capacity earlier than the SS does, shorter average response times are expected for the same capacity.

- Choosing $T_s = T_1$ is not adequate as it may happen that this point in time corresponds to a low peak in the utilisation of the task. Due to the fact that the utilisation (response time) is non monotonic with respect to server period then it may be the case that with a shorter period, higher capacity and therefore better response times can be achieved.

- Both servers can achieve very similar performance, although this happens with different server parameter configurations.

We performed several simulations like this one for different task sets with different utilisations. For all of them similar results were found.

Regarding the utilisation of the server, for low periodic utilisations, more capacity can be allocated to the task and therefore the differences in response times at different capacities are larger, anyway both servers had a similar minimum. Under high periodic loads there is almost no space for a server and in this situations small differences in server sizes were significant. However, in this case, both servers perform close to background processing which is of the order of 1 and 2 orders of magnitude larger than the requested computation time. Under this situations, no server could lead to good performance and the whole concept of using any server under such circumstances is questionable.

Regarding aperiodic computation times, the shorter the average computation time the smother the results are and the less impact there is on differences in server capacity. The larger the average computation times, the more capacity exhaustions occur and therefore the graph of the average response time is less smooth.

An additional conclusion is that queuing network models that were used in [15] and others to predict the performance of the servers are not applicable because they do not model accurately enough their behaviour. Queuing models are based on the server utilisation as the single parameter for determining the performance. We have shown that there is a huge variability on the performance of the server even for the same server utilisation. Only simulation studies have resulted is satisfactory performance predictions.

As a summary, the performance of the server depends on large server capacities and both DS and SS perform similarly although with different server parameter configurations.

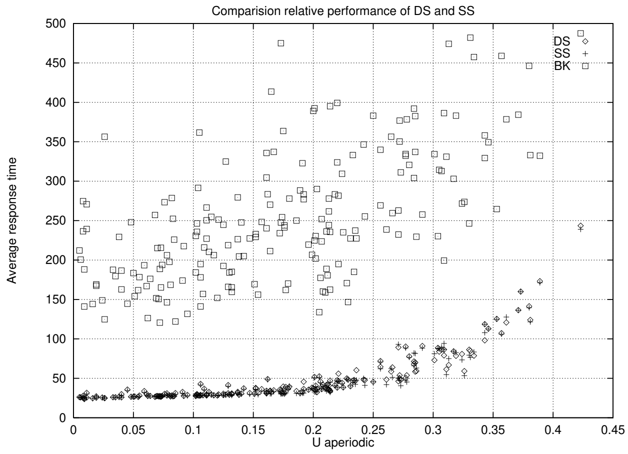

Figure 5. Average response time of aperiodic servers. ($U_p = 0.5$, $\bar{C} = 25$, $\bar{U} \in [0.01, 0.45]$)

4.3.2 DS/SS comparison

The previous study gives an indication that both servers exhibit a similar performance when their parameters are adequately configured. However, a definitive comparison of the performance of both servers is required. For this purpose we set up an experiment that tested the performance of both servers for a huge range of task sets and for a wide variety of aperiodic load conditions.

For this experiment we used groups of 200 randomly generated task sets, all task sets in the same group had the same periodic utilisation. For each task set we applied the procedure described in the following section to determine an optimal server parameter selection for both DS and SS. We then simulated the task set for a random aperiodic load in the range of $[0.01, 0.9U_a]$ where U_a is the maximum load of the server. For values larger than $0.9U_a$ the server becomes saturated and has a performance close to background service. We tested several interarrival times and mean computation times. We simulated all task sets for background processing, DS and SS (again, aperiodic task arrivals and requested computations times were reused and no execution of the servers at background priority was allowed). Similar results were found for all periodic utilisations, interarrival times and requested computation times.

Figure 5 shows the results of the simulations of one of such group of task sets with a periodic utilisation of 50% and average computation time of 25 time units. In this graph, the x axis is the aperiodic utilisation and the y axis is the average response time. Each task set is represented with three points, one for each of the scheduling mechanisms used. It can be seen that the graph has two clear regions. For low aperiodic utilisations, the response time of DS and SS is almost the requested computation time. Both servers perform almost identically as there is plenty of capacity to server the low load. For medium aperiodic load, the servers are occasionally saturated and therefore the average response time tends to grow. There are small differences between the two servers with SS, in more cases, slightly better than the DS one. For very large aperiodic utilisations the performance of the servers degrade to the performance of background scheduling. Background scheduling performs very badly compared to the servers.

We also performed the same simulations following the method described in [9] which selects $T_s = T_1$ and finds the maximum capacity c. We found that the average response times were larger than the previous simulations, which is mainly due to the smaller capacities this constraint imposes.

An additional important conclusion that stems from this experiment is that there is a large variability on the performance of the servers for different task sets. It is therefore very difficult to ascertain that certain properties will be maintained for any possible task set. Careful designed simulation studies should be performed to test the robustness of particular parameter selection.

4.4. Server parameter selection

As has been outlined before, a key for obtaining good performance of the server is based on adequate parameter configuration. From the previous analysis, it is clear that:

- Server utilisation depends on the selected capacity
- Better performance is achieved with large capacities
- Server period may be larger than the period of the highest priority hard task (server not in deadline monotonic priority ordering)

From the analysis of hundreds of task sets, we have realised that, apart from some exceptions, the lowest average response time for different aperiodic load $\bar{U}_s$ and different aperiodic computation time $\bar{C}_s$ happens at very large server capacities. The best performance can be generally achieved by selecting the capacity that corresponds to the local maxima of $U_s(c)$ closer to the maximum possible capacity.

Although there are more task sets for which the SS has a better performance than the DS, the differences are not significant. We have not considered in the simulations the overheads of the servers itself. We presume that they will produce a performance penalty, specially for the SS. Larger server capacities require multiple pending replenishments. For large server capacities the number of pending may require significant amount of memory and the scheduling algorithm that handles them will incur in additional overheads. Also it may be difficult to determine an adequate maximum on the number of pending replenishments that will also result in a poorer performance for the SS.

5. Aperiodic servers: a critique

The previous studies on aperiodic servers that have been reviewed have relied on some assumptions which are not exactly correct.

The first one is that the assumption that the SS has a higher utilisation than the DS. The utilisation based tests lead to different maximum utilisations however the exact schedulability tests have shown that the differences is utilisation are very small.

The simulation studies made some assumptions and simplifications that lead to inaccurate conclusions. First of all, selecting the maximum server period equal to the shortest period of the hard tasks resulted in lower capacities than the maximum attainable. This also resulted in an unfair parameter assignment for the DS.

The simulations also considered $\bar{C}_s$ smaller than the available capacity C_s. This resulted in analysis that did not exploit the double hit property of the DS which leads to good performance due to the early replenishments. Also very few task sets were tested and the differences between different task sets where not quantified.

Their conclusion that the SS is better than DS was drawn mainly on the fact that the maximum utilisation of the SS was larger than the DS. However, this is not significant as we have shown that the parameter that affects more the responsiveness of the server is the capacity rather than the server utilisation and both servers show similar performance at their maximum capacity even though their maximums are different.

Finally, the parameter selection technique used was not fair and lead to lower capacities and utilisations of the DS than the maximum one.

Also the current model assumptions that $D_i = T_i$ is too restrictive (this does not affect the comparison between DS and SS) as in most systems it is necessary for the deadline to be considerably smaller than the period [25]. This introduces the issue of selecting the adequate priority of the server. It is not clear that when $D_i \leq T_i$ the best performance of the server will be always achieved with a server at the highest priority.

6. Conclusion

Previous studies on aperiodic servers relied on the fact that the SS had larger utilisation and larger capacities to conclude that it had a better performance than the DS. We have shown that even though the DS has smaller capacities, it has a comparable responsiveness due to its earlier replenishment mechanism.

We have also shown that the key for good responsiveness relies on adequate server parameter selection which is, basically, selecting the largest capacity that corresponds to a local maxima of $U_s(c)$. This results in server periods which are not in deadline monotonic (nor rate monotonic) priority assignment, but that still guarantee the schedulability of the rest of the tasks in the system.

The P1003.1d standard provides support for the sporadic server only. We suggest that a generic mechanism by which different types of servers could be implemented would be desirable.

Future work in this area is carried out on the analysis of systems with multiple servers and the issues regarding optimal parameter assignment. This also rises the problem of assignment of priorities to servers other than the highest.

7. Acknowledgements

This work has been partially funded by a grant from "Caixa de Balears, Sa Nostra" and from the Universitat de les Illes Balears. The authors would like to thank Michael Gonzalez-Harbour for providing the description of the flaw in the original SS definition and to Julian Proenza for its valuable comments and discussions during the preparation of this work. We also acknowledge EPSRC grant GR/K80334.

References

[1] N. Audsley, A. Burns, M. Richardson, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] G. Bernat. *Specification and Analysis of Weakly Hard Real-Time Systems*. PhD thesis, Departament de Ciències Matemàtiques i Informàtica. Universitat de les Illes Balears. Spain, 1998.

[3] G. Bernat and A. Burns. Combining (n m)-hard deadlines with dual priority scheduling. In *18th IEEE Real-Time systems symposium. San Francisco, CA*, December 1997.

[4] A. Burns. Scheduling hard real-time systems: A review. *Software Engineering Journal*, 6(3), 1991.

[5] A. Burns and A. Welligns. Dual priority assignment: A practical method for increasing processor utilisation. Technical report, Department of Computer Science, University of York, 1993.

[6] G. Buttazzo and M. Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Transactions on Software Engineering*, 25(1):22–32, January 1999.

[7] R. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proc. 14th IEEE Real-Time Systems Symposium*, pages 222–231, North Carolina, USA, December 1993.

[8] R. Davis and A. Wellings. Dual priority scheduling. In *proc. 16th IEEE Real-Time Systems Symposium*, pages 100–109, December 1995.

[9] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, pages 31–67, 1995.

[10] IEEE Std.1003.1. Information technology - portable operating system interface (POSIX): Part 1 : System application program interface (API) [C language]. IEEE.p1003.1 1-55937-061-0, 1990.

[11] IEEE Std.1003.1b-1993. Information Technology - Portable Operating System Interface (POSIX)- Part 1: System Application program interface (API) – Amendment 1: Realtime extension. [C Language], 1993.

[12] IEEE Std.1003.1c-1995. Information Technology - Portable Operating System Interface (POSIX): Part 1 : System Application program interface (API) – Amendment 2: Threads Extension [C Language], 1995.

[13] IEEE Std.1003.1d D12 Jan 1999. Draft Information Technology - Portable Operating System Interface (POSIX): Part 1 : System Application program interface (API) [C Language]– Amendment d: Additional Realtime Extensions. [C Language], 1999.

[14] IEEE Std.1003.1i-1993. Information Technology - Portable Operating System Interface (POSIX): Part 1 : System Application program interface (API)– Amendment 3: Technical corrigenda to Realtime Extension. [C Language], 1995.

[15] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Harbour. *A Practitioner's Handbook for Real-Time Analysis: A Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[16] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks fixed priority preemptive systems. In *proc. 13th IEEE Real-Time Systems Symposium*, pages 110–123, Phoenix, Arizona, December 1992.

[17] J. P. Lehoczky, L. Sha, and J. Strosnider. Enhanced aperiodic responsiveness in hard real-time environment. In *proc. 8th IEEE Real-Time Systems Symposium*, pages 110–123, San José. California, December 1987.

[18] S. Ramos-Thuel and P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems. In *proc. 14th IEEE Real-Time Systems Symposium*, pages 160–171, North Carolina, USA, December 1993.

[19] B. Sprunt. *Aperiodic task scheduling for real-time systems*. Ph.D. thesis, Carnegie Mellon University, 1990.

[20] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *proc. 9th IEEE Real-Time Systems Symposium*, December 1988.

[21] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1:27–60, 1989.

[22] M. Spuri, G. Buttazzo, and F. Sensini. Robust aperiodic scheduling under dynamic priority systems. In *16th IEEE Real-Time Systems Symposium. Pisa, Italy*, pages 210–219, December 1995.

[23] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling for real-time systems: EDF and Related Algorithms*. Kluwer academic publishers, 1998.

[24] J. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhaced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[25] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems*, 14:219–250, 1998.