Aperiodic Servers In A Deadline Scheduling Environment

T.M. Ghazalie
T.P. Baker
Department of Computer Science
Florida State University
Tallahassee, FL 32306

October 20, 1994

Abstract

A real-time system may have tasks with soft deadlines, as well as hard deadlines. While earliest-deadline-first scheduling is effective for hard-deadline tasks, applying it to soft-deadline tasks may waste schedulable processor capacity or sacrifice average response time. Better average response time may be obtained, while still guaranteeing hard deadlines, with an aperiodic server. Three scheduling algorithms for aperiodic servers are described, and schedulability tests are derived for them. A simulation provides performance data for these three algorithms on random aperiodic tasks. The performances of the deadline aperiodic servers are compared with those of several alternatives, including background service, a deadline polling server, and rate-monotonic servers, and with estimates based on the M/M/1 queueing model. This adds to the evidence in support of deadline scheduling, *versus* fixed priority scheduling.

1 Introduction

We consider a scheduling problem in which a single processor executes a set of periodic and aperiodic tasks. The tasks are independent, except for mutual exclusion constraints over certain critical sections. A task may have a *hard* deadline, a *soft* deadline, both forms of deadline, or no deadline at all. A hard deadline is one that must always be satisfied, whereas a soft deadline is one that only represents the desired average response time. If a task has both hard and soft deadlines, the soft deadline is shorter than the hard deadline. Our primary focus will be obtaining good average response time for tasks with soft deadlines, while still being able to guarantee that all hard deadlines will be satisfied.

A hard-deadline task τ_i is a sequence of requests for execution of a particular computation, characterized by:

1. D_i, the (hard) deadline, relative to the request time, with $1 \leq C_i \leq D_i$. If execution of τ_i is requested at time t, it must be completed by time $t + D_i$. We assume the hard-deadline tasks $\tau_1, \ldots, \tau_n$ are indexed in order of increasing relative deadline, so $D_1 \leq D_2 \leq \cdots \leq D_n$.

2. C_i, an upper bound on the compute time, as a count of discrete time units.

3. T_i, an upper bound on the interarrival time, such that $1 \leq C_i \leq T_i$. We call this the "period" of the task, whether the arrival of execution requests is strictly periodic, or just sporadic.

Any task may have a soft deadline. This includes hard-deadline tasks, but also tasks for which no upper bound is known on the compute time, or no lower bound is known on the interarrival time. If a task has a soft deadline, we assume enough is known about the probability distribution of the compute times and interarrival times to speak meaningfully about average response times. For our simulations, we specifically consider tasks whose compute times and interarrival times are exponentially distributed, with known mean values.

Given a task system and arrival times, a scheduling algorithm defines a schedule, which is a conceptual time-line showing when each task executes. A schedule is said to be feasible if the deadlines of all the requests of each task are met. A task system is said to be feasible if there exists a feasible schedule for the task system.

The function of scheduling is to manage all resources to meet the timing requirements of the system. In previous work, a scheduling method has been termed "optimal" if it can find a feasible schedule for every feasible task system, but in practice this is not all that is desired. Following Sprunt, Sha, and Lehoczky [11], we recognize the following objectives:

- Guarantee tasks with hard deadlines will always complete on time.
- Achieve fast average response times for aperiodic tasks with soft deadlines.
- Attain a high degree of schedulable processor utilization.

Liu and Layland[9] showed that preemptive earliest-deadline-first (EDF) scheduling is optimal for periodic (hard deadline) task sets, under certain simplifying assumptions. They also showed that rate monotone (shortest-period-first) scheduling is optimal among static priority assignments, and they provided simple schedulability conditions for these two scheduling methods.

Other researchers have extended the applicability of these results, relaxing some of the simplifying assumptions about task arrivals, execution times, and interactions. Leung and Merrill[8] observed that if relative deadlines are distinct from periods, "rate monotone" analysis requires scheduling shortest-relative-deadline-first. Sha, Rajkumar, and Lehoczky [10] showed that the schedulability results for rate monotone priorities can be adapted to tolerate bounded blocking, such as may be due to scheduling exclusive access to shared data. These results have been extended to EDF scheduling, by Chen and Lin [2] and by Baker [1].

Sprunt, Sha, and Lehoczky [11] and Lehoczky, Sha, and Strosnider[7, 12] further extended the rate monotonic analysis model to include techniques for scheduling aperiodic tasks with soft deadlines in a fixed priority framework.

Since the Liu and Layland results show that EDF scheduling can achieve a higher degree of schedulable processor utilization than the rate monotone (fixed priority) model, for hard deadline tasks, it is interesting to see how well soft-deadline aperiodic tasks can be scheduled in an EDF framework.

In this paper, we show how to schedule soft-deadline tasks in a way that is easy to compute, does not hurt the schedulability of hard-deadline tasks, and provides good average response time for soft-deadline aperiodic tasks. We define three new algorithms, each with a sufficient test for schedulability. The schedulability tests show that deadline scheduling allows a larger server size than rate monotone scheduling.

We then report performance measurements for these algorithms, and for fixed-priority aperiodic servers, obtained via simulation. The simulation results show that the deadline servers perform much better than the fixed-priority servers, but the differences between the deadline servers are relatively small. The simulations also relate the average response time of a deadline aperiodic server to that of a simple queueing model.

The rest of the paper is organized as follows: Section 2 introduces the problem of scheduling aperiodic tasks with soft deadlines in an EDF environment. Sections 3, 4, and 5 describe new algorithms to handle this problem, and derive schedulability tests for these algorithms. Section 6 compares these algorithms on an example. Section 7 explains how these algorithms can be combined with background processing. Section 8 reports on a simulation study of the performance of the three aperiodic server algorithms, comparing them against some previously known algorithms and against queueing models. Section 9 reports results of a comparsion study with rate monotone scheduling and fixed-priority aperiodic servers. Section 10 concludes the paper.

2 Scheduling For Soft Deadlines

Tasks with soft deadlines pose a problem for EDF scheduling. The basic problem is that the EDF scheduler cannot tell a soft deadline from a hard deadline. With pure EDF scheduling, the only way to improve the response time of a task is to shorten its deadline. Since the EDF scheduler will then treat this soft deadline as a hard deadline, it can cause truly hard deadlines of other tasks to be missed.

The impact of soft deadline aperiodic requests on tasks with hard deadlines can be reduced by using an *aperiodic server* task. Aperiodic requests are queued for the server when they arrive, and executed as soon as the scheduling algorithm permits the server to execute them.

A simple form of aperiodic server is a *background* server. The server is scheduled at lower priority than every hard deadline task. In this way, soft deadline requests never cause a hard deadline to be missed. The main drawback of this model is that both average and worst-case response times for the server may be unacceptably long.

Another simple form of aperiodic server is a *polling server*. With polling, the server is treated as a hard deadline periodic task with a fixed execution time budget, whose deadline is equal to its period. Once each period, the server executes and serves all the requests that have been enqueued up to that time. If there are more requests than can be served in the budgeted time they are carried over to the next period. If there is any left-over execution time budget it is discarded. The period is chosen short enough to achieve the desired average response time and to guarantee any associated hard deadline. The queueing of requests allows the period of the server to be longer than the interarrival time of the requests, so that the adverse effect on schedulability of other tasks is reduced.

An advantage of polling over background processing is that hard deadlines (of tasks that have both hard and soft deadlines and have bounded execution and interarrival times) can be guaranteed, since the server periodp is treated as a hard deadline. Also, by using multiple servers, at different priority levels, one can accomodate a set of tasks with a range of hard and soft deadline requirements.

The main disadvantage of polling service is that, under reasonable assumptions about the distribution of aperiodic requests, the average response time is at least half the server period plus the average execution time. Thus, the only way to improve response time for soft-deadline tasks is to reduce the server period. For very short server periods, polling service becomes time-sharing; that is, the response time approaches that of a dedicated processor whose speed is slowed down by a factor of c ($0 < c <= 1$), where c is the fraction of the processor allocated to the server. This behavior is shown in Figure 5. The segmented lines are average response times a simulated polling server, with 69% periodic load, for several different server periods. (The simulation is described in Section 8.) The limiting case of a time-sliced server, with infinitesimal time slicing interval and the same processor utilization as the polling server, is shown by the upper smooth line. The lower smooth line is the expected performance of a processor 100% dedicated to serving the aperiodic load, with no periodic load.

Since these simulations do not take into account the overhead of frequent polling, the performance is better than it would be in an application. This overhead of polling goes up as the polling period gets shorter, and is the real limitation on the feasible frequency of polling.

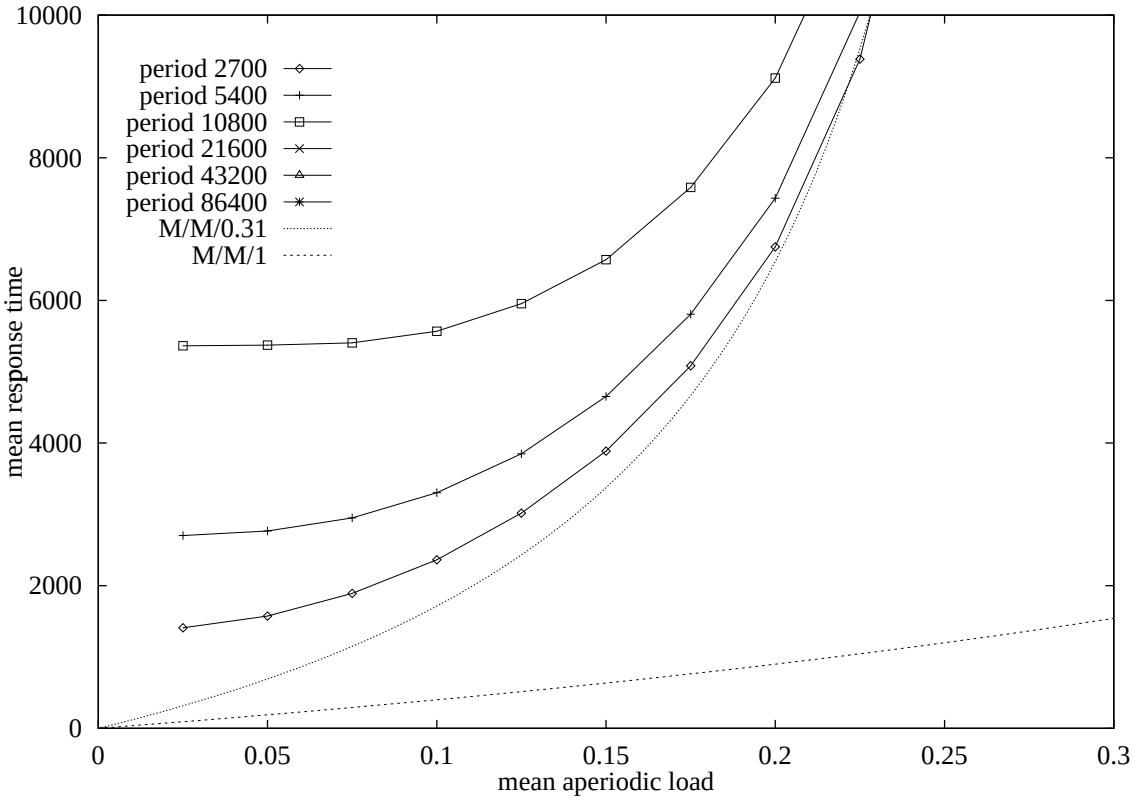

Figure 1: Polling Server response time versus period, IAT 3600, 69% periodic load.

We will show that better scheduling techniques are possible, that provide both shorter response times for aperiodic tasks with soft deadlines than time-slicing, and lower overhead, while still guaranteeing hard deadlines. In the following three subsections, we will describe three such techniques.

3 Deadline Deferrable Server Algorithm

Lehoczky, Sha, and Strosnider in [7, 12] describe an approach which can be used to give better response time for aperiodic tasks within the framework of rate monotone scheduling. They call this the *deferrable server* (DS). The DS algorithm, like a polling server, creates a periodic task (usually of high-priority) for servicing aperiodic requests. However, unlike

polling, the algorithm preserves the execution time allocated for aperiodic service if, upon the invocation of the server task, no aperiodic requests are pending. The DS algorithm maintains the aperiodic server's execution time budget for the current period, as long as it has not been exhausted. At the beginning of the next DS period, the server's high-priority execution time budget is replenished to its full size.

Though Lehoczky, Sha, and Strosnider proposed the DS for use with fixed priority scheduling, we will show how this idea can be adapted to EDF scheduling. To do this, we need to modify how the server preserves its high-priority execution time. We call the resulting new server model the *deadline deferrable server* (DDS).

The deadline deferrable server is a periodic task whose relative deadline is the same as its period, T_S. As with the DS, the difference between the DDS and a polling server is that the DDS is able to serve requests that come in during the middle of a period. At the beginning of each period, it receives an execution time budget allocation of a fixed size, C_S. If, after any queued requests are served, the execution time budget is not exhausted, the remaining portion of the budget is retained up to the end of the period, and can be used for requests that come in during that time. So long as a portion of this budget remains, the server is allowed to execute at the priority determined by its deadline. When the budget is consumed, the server is suspended until the next period.

The net effect of the DDS scheduling policy is that, independent of the actual arrival time, an aperiodic request will be served at a priority consistent with the request having come in at the beginning of the period, so long as the server's execution time budget has not been exhausted.

The analysis of the effect of the DDS on schedulability of other tasks is based on the following lemma, which gives the worst case of the aperiodic server's demand for CPU time in a time interval $[t', t]$.

Lemma 1 *For every time interval $[t', t]$ such that t is a missed deadline, $t' \leq t - C_S$, and there is no task with deadline later than t executing in the interval, an upper bound of the deadline deferrable server's demand for CPU time in the interval is*

$$\left\lfloor \frac{\Delta - C_S}{T_S} \right\rfloor C_S + C_S$$

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

Proof:

Consider the situation in Figure 2. This is a worst case. It occurs if an aperiodic request with execution time C_S arrives C_S time units before the end of a server period and there is a

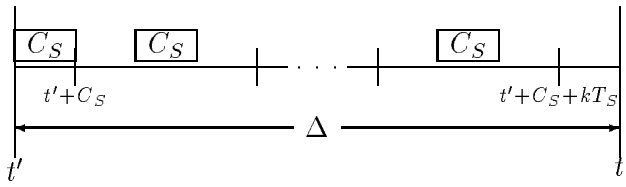

Figure 2: Worst case situation

similar aperiodic request at every later server period. Because the DDS priority is retained, by preserving the server's deadline, the first aperiodic request will have the shortest deadline; we service this request of the aperiodic task immediately, which uses C_S time units.

If $t = t' + C_S + (k+1)T_S$, the aperiodic server demand for CPU time in $[t' + C_S, t]$ is

$$\left(\frac{\Delta - C_S}{T_S}\right) C_S = \left\lfloor \frac{\Delta - C_S}{T_S} \right\rfloor C_S.$$

Otherwise, if $t < t' + C_S + (k+1)T_S$, the aperiodic server demand for CPU time in $[t' + C_S, t]$ is

$$\left\lfloor \frac{\Delta - C_S}{T_S} \right\rfloor C_S$$

because t is an earlier deadline that is missed, and so under the EDF scheduling algorithm, the aperiodic server will not execute between $t' + C_S + kT_S$ and t.

It follows that the total demand for CPU time by the aperiodic server in $[t', t]$ is at most

$$\left\lfloor \frac{\Delta - C_S}{T_S} \right\rfloor C_S + C_S$$

□

The following theorem establishes a sufficient condition for feasibility of a set of (periodic and aperiodic) tasks under the DDS algorithm. This theorem is based on the schedulability condition for EDF scheduling derived by Baker in [1], which is a generalization (to include blocking times and aperiodic tasks, and to relax the requirement that relative deadline equals period) of the schedulabilty test for deadline-driven scheduling originally derived by Liu and Layland in [9]. Here, for simplicity of presentation, the effects of critical sections in both the server and other tasks are ignored.

Theorem 2 *A set of (periodic or aperiodic) hard-deadline tasks is schedulable by EDF scheduling and DDS algorithms if*

$$\forall_{k=1,\ldots,n} \quad \left(\sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)}\right) + \left(1 + \frac{T_S - C_S}{D_k}\right) U_S \leq 1$$

where $U_S = C_S/T_S$ is the aperiodic server utilization.

Proof:

The proof is based on assuming that some task τ_m can miss its deadline, examining a busy interval $[t', t]$ before the first point τ_m misses its deadline, summing up the worst-case execution times of all the tasks that may preempt τ_m in such an interval, and dividing this by the shortest possible length of the interval.

Let t be the first time a task τ_m misses its deadline, and t' be the last time before t such that there are no pending task execution requests with arrival times before t' and deadlines before t. Call the interval $[t', t]$ the *busy interval*.

Let k be the index of the task with the longest relative deadline, of all the tasks requested in the busy interval. It follows that the set $\mathcal{A} = \{\tau_1, ..., \tau_k\}$ includes all the tasks requested in the busy interval, other than the DDS. It also follows that every request for a task in $\mathcal{A}$ that occurs in the busy interval has higher EDF priority than any request for a task τ_b, $b > k$.

By choice of t', there are pending requests for tasks in $\mathcal{A}$ or the DDS at all times during the busy interval, so the processor will always be busy. By the EDF priority assignment, the only tasks that will be allowed to start execution in the busy interval will be the DDS and those in $\mathcal{A}$, and these can only be preempted by the DDS and other requests for tasks in $\mathcal{A}$.

Let $\Delta = t - t'$. By choice of t', $\Delta \geq D_i$ for every τ_i in $\mathcal{A}$, and in particular $\Delta \geq D_k$.

From Lemma 1, the maximum time demand of the deadline deferrable server in the busy interval is

$$\left\lfloor \frac{\Delta - C_S}{T_S} \right\rfloor C_S + C_S$$

For every hard-deadline periodic or aperiodic task τ_i, $i \leq k$, the demand for CPU time in the busy interval is not more than

$$\left(\left\lfloor \frac{\Delta - D_i}{T_i} \right\rfloor + 1 \right) C_i.$$

In the busy interval there is no idle time. The only tasks executing are the DDS, and $\tau_1, \ldots, \tau_k$. Therefore, the total demand for CPU time is at most

$$\mathcal{D} = \sum_{i=1}^{k} \left(\left\lfloor \frac{\Delta - D_i}{T_i} \right\rfloor + 1 \right) C_i + \left(\left\lfloor \frac{\Delta - C_S}{T_S} \right\rfloor + 1 \right) C_S$$

Since there is an overflow, the total demand for CPU time in the busy interval exceeds Δ. It follows that if τ_m misses its deadline, then $\mathcal{D} > \Delta$, and so $\mathcal{D}/\Delta > 1$.

Since $\lfloor x \rfloor \leq x$,

$$\frac{\mathcal{D}}{\Delta} \leq \mathcal{L}_0 = \sum_{i=1}^{k} \left(\frac{\Delta - D_i}{T_i} + 1 \right) \frac{C_i}{\Delta} + \left(\frac{\Delta - C_S}{T_S} + 1 \right) \frac{C_S}{\Delta}.$$

It follows that if τ_m misses its deadline then $\mathcal{L}_0 > 1$.

If $D_i \leq T_i$

$$\left(\frac{\Delta - D_i}{T_i} + 1\right)\frac{C_i}{\Delta} \leq \left(\frac{\Delta - D_i}{D_i} + 1\right)\frac{C_i}{\Delta} = \frac{C_i}{D_i}$$

and if $D_i > T_i$

$$\left(\frac{\Delta - D_i}{T_i} + 1\right)\frac{C_i}{\Delta} < \frac{C_i}{T_i}$$

so, in general

$$\left(\frac{\Delta - D_i}{T_i} + 1\right)\frac{C_i}{\Delta} \leq \frac{C_i}{\min(D_i, T_i)}$$

It follows that

$$\begin{aligned}\mathcal{L}_0 \leq \mathcal{L}_1 &= \sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)} + \left(\frac{\Delta - C_S}{T_S} + 1\right)\frac{C_S}{\Delta} \\ &= \sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)} + \left(1 + \frac{T_S - C_S}{\Delta}\right) U_S\end{aligned}$$

Since $D_i \leq \Delta$ for $i = 1, \ldots, k$

$$\mathcal{L}_1 \leq \mathcal{L}_2 = \sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)} + \left(1 + \frac{T_S - C_S}{D_k}\right) U_S$$

Thus, if τ_m misses its deadline then $\mathcal{L}_2 \geq \mathcal{L}_1 \geq \mathcal{L}_0 > 1$.□

The theorem and proof are stated above for only one server, but the reasoning of the proof of Lemma 1 remains valid if there are other servers. Thus, Theorem 2 can be generalized to multiple servers. Dividing the aperiodic load between several servers can be helpful if there are aperiodic tasks with different hard timing constraints. As will be shown in Section 8, a longer server period may reduce the average response time, up to a certain point. However, the effect of back-to-back server executions, reflected in the term $(T_S - C_S)/D_k$) of Theorem 2, requires a reduction of the server utilization for large server periods to prevent missed periodic deadlines. This relationship is illustrated in Figure 3, for three sample periodic task sets. (These task sets are described completely in Section 8.)

It is left as an exercise for the reader to verify that the theorem can also be generalized to take into account critical sections. Suppose nonpreemptable resources are allocated in a way that bounds priority inversion, such as the SRP (Stack Resource Protocol[1]). If the server time budget is exhausted during a critical section, the server may overrun its budget by the duration of that critical section. To ensure that this effect cannot carry over to the next execution of the server, the amount of execution time allocated to the server at the beginning

of each period needs to be reduced by the amount of overrun in the previous period. With terms added for blocking effects, the schedulability condition of the theorem then becomes:

$$\forall_{k=1,\ldots,n} \quad \left(\sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)}\right) + \frac{B_k + \epsilon_S}{D_k} + \left(1 + \frac{T_S - C_S}{D_k}\right) U_S \leq 1$$

where $U_S = C_S/T_S$ is the aperiodic server utilization, ϵ_S is the execution time of the server's longest critical section, $B_k = \max\{\epsilon_i \mid i < k\}$, and ϵ_i is the execution time of the longest critical section of the task τ_i.

With the SRP, because an arriving request may cause the deferrable server to preempt with arbitrarily short deadline, the ceilings of resources used by the server must be set to the maximum. That is, server critical sections cannot be preempted by any other task.

4 Deadline Sporadic Server Algorithm

Sprunt, Sha, and Lehoczky in [11] describe another approach to scheduling an aperiodic server within the framework of rate monotone scheduling. They call this the *sporadic server* (SS). Like the DS, the SS is a high-priority task that services aperiodic tasks. The first difference is that the sporadic server can preserve its unused high-priority execution time indefinitely. The other difference is that the replenishment of execution time used by the SS is scheduled in a way that forces the execution to be spread out more evenly; in particular, the worst-case situation of the DS, where two server allotments are used back-to-back, cannot occur. A key property of the SS is that its effect on the schedulability of tasks with lower priorities cannot be worse than that of a periodic task with the same period and execution time equal to the server size. In this respect the SS is an improvement over the DS.

As with the DS, we will adapt the SS from fixed priority to EDF scheduling. The basic idea of the SS will be retained: the delay until replenishment of the server's execution time budget will be measured from when the server priority becomes active. The modification will be in the assignment of priority, through an appropriate choice of deadline. We call the resulting new server model the *deadline sporadic server* (DSS).

A deadline sporadic server has a period T_S, and an execution time C_S. The DSS scheduling algorithm attempts to budget the server's execution time in such a way that the effect of the server on the schedulability of hard-deadline tasks is no worse than that of a hard-deadline periodic task with period and deadline T_S, and execution time C_S.

When requests for the DSS come in, they are put onto the server queue. The arrival time of each request is also recorded on the server queue, with the request. The server executes

requests from the server queue in order of arrival, as soon as it is permitted to do so by the server scheduling algorithm.

The DSS scheduling algorithm keeps track of the server execution time budget in *chunks*. At any instant, the server's budget is partitioned into one or more chunks $\{\chi_1, ..., \chi_n\}$. The scheduling algorithm splits and merges chunks, so the actual set and number of chunks varies over time. Each chunk has an associated *size*, $\sigma_i \geq 1$. The sum of the sizes of all the chunks is constant, $C_S = \sigma_1 + \cdots + \sigma_n$. Each chunk has an associated *replenishment time*, ρ_i. If the replenishment time is in the future, the chunk is *awaiting replenishment*; otherwise it is *available*. Initially, there is one chunk, and it is available. The chunk's size is C_S and its replenishment time is the system start time.

The server is only *eligible to execute* when there are one or more uncompleted requests on the server queue, and at least one available chunk of execution time. Let χ_m always denote the available chunk with the earliest replenishment time. The execution time used by the server is charged to χ_m.

When the server consumes all of the chunk χ_m or completes all the queued requests, the portion of χ_m that the server has consumed is split off and scheduled for replenishment. Suppose the server has used δ execution time. A new chunk, $\chi_{m'}$, is created, with size $\sigma_{m'} = \delta$ and replenishment time $\rho_{m'} = d_S$, where d_S is the current deadline of the server (which is defined further below). The size of χ_m is reduced by δ. If the new size is zero, the chunk χ_m is deleted, and its role is then played by the remaining chunk with earliest replenishment time.

For scheduling purposes, the server has a *deadline*. This deadline is $t_z + T_S$, where t_z is defined as follows:

1. Initially, t_z is undefined.
2. If t_z is undefined, and the server becomes eligible to execute, t_z is set to $Clock$.
3. If t_z is undefined, and a task with deadline $d \leq Clock + T_S$ starts to execute, t_z is set to $Clock$.
4. If t_z is defined, and a task with deadline d such that $t_z < d - T_S \leq Clock$ starts to execute, t_z is updated to $d - T_S$.
5. If t_z is defined, and a task with deadline d such that $Clock < d - T_S$ starts to execute, t_z becomes undefined.
6. If t_z is defined, and the server begins a chunk χ_m with $\rho_m > t_z$, t_z is updated to ρ_z.

In each case, *Clock* denotes the time at which the condition becomes true. Assume there is an idle-task, with deadline ∞, that executes when no other task is eligible to execute; by the fifth case above, t_z is undefined when the idle task is running.

Stated informally, t_z is *the most recent time at which the (current) server priority became active*. However, we are forced define it without reference to the server priority, to avoid circularity in the definition, since the priority depends on the deadline, which depends on t_z.

Note that t_z is always defined when the server is eligible to execute, and is never later than the time the server last became eligible to to execute. Sometimes it may be earlier, if the processor was continuously occupied with higher priority tasks just before the server became eligible to execute.

For any time t when t_z is defined, it is true that in the interval $[t_z, t]$ the processor is continuously busy executing tasks with deadlines $\leq t_z + T_S$. It follows that the effect on the schedulability of other tasks is the same as if the server had become eligible to execute at t_z and remained eligible to execute during the entire interval $[t_z, t]$.

Note that the replenishment method of the DSS algorithm forces the server not to re-use a chunk of execution time until at least T_S time units later than the server last could have become eligible to use that chunk. Thus, the execution time of the server within any time interval cannot exceed the amount that would be used by a periodic task with the same period and execution time.

With the DSS scheduling algorithm as we have just described it, the execution time budget will become fragmented into smaller and smaller chunks. This fragmentation can be limited by a simple modification. Available chunks are coalesced when this can be done without affecting the scheduling outcome. If t is any time the server priority is inactive (or the instant where the server priority becomes active), at time t we can coalesce all the available chunks into a single chunk with replenishment time t. This cannot affect the scheduling outcome, since ρ_i is only used in updating t_z (and computing the server deadline) when $\rho_i > t_z$, and we know $t_z \geq t$.

Based on the arguments above, we have the following lemma, which characterizes the deadline sporadic server load.

Lemma 3 *For every time interval $[t', t]$ such that t is a missed hard deadline, $t' \leq t - C_S$ and there is no task with deadline later than t executing in the interval, an upper bound of the deadline sporadic server's demand for CPU time in the interval is*

$$\left\lfloor \frac{\Delta}{T_S} \right\rfloor C_S$$

where $\Delta = t - t'$.

The following theorem provides a sufficient condition for feasibility of a set of (periodic and aperiodic) tasks under the DSS algorithm.

Theorem 4 *A set of hard-deadline tasks is schedulable by EDF scheduling and the DSS algorithm if*

$$\forall k \quad \left(\sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)} \right) + U_S \leq 1,$$

where $U_S = C_S/T_S$ is the aperiodic server utilization.

Proof:

The proof is similar to the proof of Theorem 2. The total demand for CPU time in the busy interval is at most

$$\sum_{i=1}^{k} \left(\left\lfloor \frac{\Delta - D_i}{T_i} \right\rfloor + 1 \right) C_i + \left\lfloor \frac{\Delta}{T_S} \right\rfloor C_S$$

Since there is an overflow, the total demand for CPU time in $[t', t]$ exceeds Δ, so

$$\sum_{i=1}^{k} \left(\left\lfloor \frac{\Delta - D_i}{T_i} \right\rfloor + 1 \right) C_i + \left\lfloor \frac{\Delta}{T_S} \right\rfloor C_S > \Delta.$$

By similar reasoning to the proof of Theorem 2, we obtain

$$\sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)} + \frac{C_S}{T_S} > 1.$$

□

The theorem and proof above are stated for a single server but can be generalized to multiple servers, with different sizes and replenishment periods. For example, it can be seen that the schedulability test is satisfied for two servers with sizes C_S and $C_{S'}$ and periods T_S and $T_{S'}$, respectively, if-and-only-if it is satisfied for a single server with utilization $\frac{C_S}{T_S} + \frac{C_{S'}}{T_{S'}}$. As shown in Section 8, in the context of a given periodic task load, the average response time of aperiodic tasks may be improved significantly by assigning a longer server period. This pays off better with the sporadic server than the deferrable server, since the server's impact on periodic tasks depends only on the server utilization; that is, one can adjust the server period and server size freely, without causing missed periodic deadlines, so long as the server utilization remains constant.

It is left as an exercise for the reader to verify that the schedulability condition of the Theorem 4 can be generalized to account for the blocking effects of critical sections. The replenishment must be adapted to prevent server overruns from accumulating. If the server uses up a chunk while it is in a critical section, it is allowed to continue executing at the same priority until it leaves the critical section. This is an execution time *overrun*. When

the server exits an outermost critical section, the scheduler must check whether the current chunk has been exhausted. If the chunk has been exhausted, the scheduler does the normal processing for the replenishment of that chunk, but keeps out the amount of the execution time overrun. This will be charged against future replenishments in a way that mimics what would have happened if the task had not been allowed to run over.

Suppose the amount of the server overrun is x. Suppose the next replenishment is χ_m, which arrives at time t. There are two possible cases:

1. $x \geq \sigma_m$. The overrun amount is reduced to $x - \sigma_m$, and the replenishment time of χ_m is updated to $t + x + T_S$.

2. $x < \sigma_m$. The overrun amount is reduced to zero, and the size of χ_m is reduced by x.

The schedulability condition with blocking terms is then

$$\underset{k=1,\ldots,n}{\forall k} \quad \left(\sum_{i=1}^{k} \frac{C_i}{\min(D_i, T_i)} \right) + \frac{B_k + \epsilon_S}{D_k} + U_S \leq 1$$

With the DSS, unlike the DDS, an arriving aperiodic request cannot cause the server to preempt a periodic task that has a shorter relative deadline than the server period. Thus, the server contribution to SRP ceilings is determined by the server period. If there are other tasks with shorter deadlines than the server period, that do not lock any resources used by the server, they are not blocked by server critical sections. This reduced blocking can be an advantage for the DSS.

5 Deadline Exchange Server Algorithm

The third algorithm is a modification and simplification of the DSS algorithm. The DSS algorithm requires keeping track of an arbitrary number of replenishment chunks, with various sizes and replenishment times. This makes the implementation complicated, especially if one wants to support several servers with different capacities and priority levels. Looking for a simpler algorithm with similar performance to the DSS, we came up what we call the *Deadline Exchange Server* (DXS). The idea of the DXS is to discard the remaining server execution time as soon as the request queue is empty in exchange for earlier replenishment of the full server budget.

The operation of the DXS algorithm is similar to the DSS algorithm, except that any remaining server execution time is discarded as soon as the request queue is empty, and the wait for replenishment time is proportional to the size of the last chunk of execution time used. That is, if we only used x time units of server execution time out of the server size C_S,

then instead of replenishing this x time units at $t_z + T_S$, we discard the remaining execution time and schedule a full replenishment of C_S units to occur at time $t_z + \frac{x}{C_S}T_S$. The server deadline is computed the same as for the DSS, i.e. it is still $t_z + T_S$.

The analysis of the DXS algorithm is simplified by the fact that (unlike the DSS) the DXS budget is always zero when it is waiting for a replenishment. Thus, at all times, the DXS must be in exactly one of the following states.

1. The server is not executing, there are no pending requests, and the server budget is full.
2. The server is not executing and the server budget is empty. (There may or may not be pending requests.)
3. The server is executing, there is at least one pending request (which is being served), and the server budget is non-empty.

This case breakdown is used to proove the following lemma.

Lemma 5 *For every time interval $[t', t]$ such that t is a missed hard deadline, $t' \leq t - C_S$, and there is no task with deadline later than t executing in the interval, an upper bound on the deadline exchange server's demand for CPU time in the interval is*

i. Δ, *for* $\Delta < C_S$

ii. C_S, *for* $C_S \leq \Delta < T_S$

iii. $\frac{C_S}{T_S}\Delta$, *for* $T_S \leq \Delta$

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

Proof:

The proof is by induction on Δ. For $\Delta \leq C_S$, it is clear that *(i)* holds, since the server cannot execute longer than the length of the entire interval. This is the basis of the induction.

Consider an interval $[t', t]$ as in the statement of the lemma. At time t', the server must be in one of the three states described above. Consider what happens in each case.

1. The server budget is full. There are no pending requests at time t'. If there is no request in the interval $[t', t]$, the server does not execute, and the upper bounds are satisfied. Otherwise, suppose the next server request arrives at time $t' + \epsilon$, $1 \leq \epsilon < \Delta$. The server time used in the interval $[t', t]$ is the same as the server time for the interval $[t' + \epsilon, t]$. There are two cases.

(a) $\Delta - \epsilon < T_S$. The server cannot execute in the interval $[t' + \epsilon, t]$, since its deadline is $t' + \epsilon + T_S$, which is later than t, and there is a periodic task with deadline t that has not completed.

(b) $T_S \leq \Delta - \epsilon$. By induction, the lemma holds for the interval $[t' + \epsilon, t]$, since it is shorter than $[t', t]$. Therefore, the server uses at most $\frac{C_S}{T_S}(\Delta - \epsilon)$ time in $[t' + \epsilon, t]$, which is less than $\frac{C_S}{T_S}\Delta$.

In both cases it follows that the upper bounds of the lemma are satisfied for $[t', t]$.

2. The server is not executing and the server budget is empty. In this case, the server cannot execute until the next replenishment time. If the replenishment does not occur within the interval $[t', t]$, the server will not be able to preempt the task whose deadline is missed at t and the upper bounds are satisfied. Otherwise, suppose the next replenishment is at time $t' + \epsilon$, $1 \leq \epsilon < \Delta$. Applying induction the same reasoning as in the previous case, we see that the upper bound holds.

3. The server is executing a request. This means the server budget is non-empty. The server will execute (subject to preemption by tasks with shorter relative deadlines) until it has served all pending requests or its budget has become empty. Let t'' be the time the server stops. The replenishment will be scheduled no earlier than time $t' + \frac{T_S}{C_S}\epsilon$, where ϵ is the amount of time used by the server in the active period ending at t''.

   We have two cases:

   (a) $t' + \frac{T_S}{C_S}\epsilon + T_S > t$. The server cannot execute in $[t' + \frac{T_S}{C_S}\epsilon, t]$, since the deadline associated with the replenishment is after t.

   (b) $t' + \frac{T_S}{C_S}\epsilon + T_S \leq t$. By induction, the lemma holds for the interval $[t' + \frac{T_S}{C_S}\epsilon, t]$, since it is shorter than $[t', t]$. Since $T_S \leq \Delta - \frac{T_S}{C_S}\epsilon$, case (iii) applies. Therefore, the server uses at most $\frac{C_S}{T_S}(\Delta - \frac{T_S}{C_S}\epsilon)$ time in $[t' + \frac{T_S}{C_S}\epsilon, t]$, which means the time used in $[t', t]$ is at most

$$\epsilon + \frac{C_S}{T_S}(\Delta - \frac{T_S}{C_S}\epsilon) = \frac{C_S}{T_S}\Delta.$$

   In all cases, the upper bound is satisfied.

□

Lemma 6 *The upper bounds of Lemma 5 are achievable.*

Proof:

Given a length, Δ, there two possible cases:

1. $\Delta < T_S$. Suppose the server budget is full at time t and an unbounded stream of requests arrives at that time. The server will execute for C_S time and then wait until $t' + T_S$ for replenishment. Thus, it uses Δ time within the interval if $\Delta < C_S$, and uses C_S time if $C_S \leq \Delta$.

2. $T_S \leq \Delta$. The maximum server load over the interval is achieved under the following scenario.

   (1) The server executes for the first C_S time units of every interval $[t - iT_S, t - (i-1)T_S]$, for $i = 1, \ldots, k$, where $k = \lfloor \frac{\Delta}{T_S} \rfloor$, and

   (2) it executes for the first $\frac{C_S}{T_S}\epsilon$ time units of the interval $[t', t' + \epsilon]$, where $\epsilon = \Delta - kT_S$, and k is as above.

   This execution pattern is achieved if the server budget at time t' is $\frac{C_S}{T_S}\epsilon$ and the rate of requests is sufficient to keep the server busy continually thereafter. The total server execution time in the interval is

$$kC_S + \frac{C_S}{T_S}(\Delta - kT_S) = \frac{C_S}{T_S}(\Delta)$$

□

The proof of Theorem 4 also applies to the DXS, except that (by Lemma 5) the floors are replaced by exact fractions. Thus, the sufficient condition for DSS schedulability also applies to the DXS.

Since the proof does not depend on whether there is more than one server, or on the relationship of the period of the server to the deadlines of other tasks, the extension to account for blocking times also applies. As with the DSS, the relative deadline of the server can be used in computing the SRP ceilings for resources used by the server.

Comparing the behavior of the DXS and the DSS, we see that neither is better than the other in all cases. For example, suppose the DSS has size C and period T. The server budget is C at time 0. Suppose a request arrives at time $T/2$ that takes $C/2$ time to execute. This leaves a server budget of $C/2$. Suppose the next request arrives at time T and needs C time to execute. The server runs for $C/2$ time, and then must wait for the replenishment that will occur at time $T + T/2$. The request finally completes at time $T + T/2 + C/2$. With DXS, the second request would have to wait, at first, since the excess server budget $C/2$ would be discarded at the end of the first request, but replenishment would occur earlier, at time T. This would enable the DXS to complete the second request by time $T + C/2$ — almost 50% earlier than the DSS. There are also situations where the DXS does worse. In particular, this would be the case if a request with large execution time were immediately

followed by one with a very short execution time. With the DXS it would be necessary to wait for replenishment before serving the short request, but with DSS there would be enough remaining execution time in the budget for the short request to be served without waiting for replenishment.

The DXS is likely to do worse than the DSS and DDS for bursty arrivals, that are close together but not close enough that one arrives while the preceding one is still being served, assuming the server size is chosen large enough to handle most bursts. When an aperiodic request arrives after the preceding request has been served to completion, the DXS will always force it to wait a while, for replenishment of the time used by the preceding request.

The chief advantage of the DXS is that it is much simpler to implement than the DSS. An implementation of the DSS must do a lot of housekeeping to keep track of the set of pending replenishments, which have different sizes and different times. In contrast, the DXS never more has than one replenishment pending at any time, and the amount of the replenishment is always the same, i.e. the full server quota. This advantage in simplicity is likely to be especially noticeable if one needs to support multiple servers, with different sizes and replenishment periods.

6 Example

The DDS, DSS, and DXS algorithms will be illustrated by comparing their operation to that of background and polling aperiodic service using a simple periodic task set. The task set is composed of two periodic tasks: τ_1 and τ_2. For these examples, both τ_1 and τ_2 begin their periods at time = 0. The periodic task set parameters are as follows:

Task	Exec. time	Period	Utilization (%)
τ_1	2	10	20.0
τ_2	6	15	40.0

For this periodic task set, the maximum server sizes were determined for the polling server, the DDS, DSS, and DXS algorithms. For the DDS, DSS, and DXS algorithms, the server sizes were determined based on the sufficient conditions derived in Theorem 2 and Theorem 4. For each algorithm, the server's period was chosen to be 5 units of time. The servers were started at time zero. The server size characteristics for the polling server, DDS, DSS, and DXS algorithms are as follows:

Algorithm	Exec. time	Period	Server utilization (%)
Polling	2	5	40.0
DDS	1.63	5	32.6
DSS	2	5	40.0
DXS	2	5	40.0

Figures 4, 5, 6, 7, and 8 show the behavior of the background service, polling service, the DDS, DSS, and DXS Algorithms for this task set. Figure 4 and the upper part of Figures 5, 6, 7, and 8 show the task execution order, and the lower part of Figures 5, 6, 7, and 8 shows the server budget as a function of time. In each of these examples, two aperiodic requests occur. The first aperiodic request for 1.8 time units and occurs at time = 2, and the second is for 2 time units and occurs at time = 6.

The response time performance of background service for aperiodic requests shown in Figure 4 is poor. Since background service only occurs when the resource is idle, the first aperiodic service begins at time = 8 and part of the second aperiodic service begins at time = 9.8 before pereempted by periodic task τ_1 and resumed at time = 12. The response times of both aperiodic requests are 7.8 time units.

The response time performance of polling service for the aperiodic requests shown in Figure 5 is better than background service for both requests—the response time of the two aperiodic requests are 4.8 and 5.8 time units respectively. The polling server's first period begins at time = 0. As can be seen on the budget graph, the execution time budget of the polling server is discarded during its first period because no aperiodic requests are pending. The first request misses the first polling period and must wait until the second polling period (time = 5) before being serviced. At time = 5, the first aperiodic request receives immediate service, because it has the nearest deadline. Part of the second request can be serviced using the remaining execution time, and it is completed in the third polling period. At time 11.8, the service of the second request is completed, and since there is no further request, the remaining execution time is discarded.

Figure 6 shows the behavior of the DDS algorithm. At time = 0, the server's execution time budget is brought to its full size of 1.63 unit and τ_1 begins execution. The server budget and deadline are preserved until the first aperiodic request occurs and is serviced at time = 2. At time = 3.63, the server's execution time is exhausted and τ_2 begins. At time = 5, the server's execution time is brought to its full size of 1.63 unit and the service for the first aperiodic request resumes, consuming 0.17 units of server execution time. At time = 5.17, the service for the first request is completed and τ_2 resumes execution. The response time for the first aperiodic request is 3.17 units of time. At time = 6, the second aperiodic request occurs and is serviced using the remaining 1.46 units of server execution time. At time = 7.46, aperiodic service is suspended and τ_2 resumes execution. At time = 10, the server's execution time is brought to its full size, 1.63. Assuming the aperiodic server wins priority ties, such as with τ_2, service for the second aperiodic request is resumed. At time =10.54, service for the second aperiodic request is completed (leaving 1.09 units of server execution time) and τ_2 resumes execution. The response time for the second aperiodic request is 4.54

units of time. At time = 12, τ_2 completes execution and τ_1 begins execution. At time = 14, τ_1 completes execution. At time = 15, the server's execution time is brought to its full size of 1.63 unit and τ_2 begins execution.

Figure 7 shows the behavior of the DSS algorithm. The server begins with its full execution size of 2.0 unit. At time = 0, τ_1 begins execution and t_z is undefined. At time = 2, τ_1 completes execution and the first aperiodic request occurs and is serviced immediately since $d_S = t_z + T_S = 7$. At time = 3.8, the servicing of the first aperiodic request is completed, leaving 0.2 of the server's execution time, and τ_2 begins execution. At this time t_z is undefined since τ_2's deadline is at time = 15. A replenishment of 1.8 units of time is set for time = 7 (denoted by arrow pointing from time = 2 on the task execution line to time = 7 on the server budget time line in Figure 7). The response time of the first aperiodic request is 1.8 units of time. At time = 6, the second aperiodic request occurs. At this time the server becomes eligible to execute, so t_z is set to 6. ince d_S is equal to 11, the request can be serviced immediately using the remaining 0.2 units of server time. At time = 6.2, the server is exhausted and the replenishment of 0.2 units of time is set for time = 11. Task τ_2 resumes its execution. At time = 7, the first replenishment of server execution time occurs, bringing the server's budget up to 1.8 units of time. t_z becomes defined, and we can resumes the execution of the second aperiodic request since d_S is equal to 12. At time = 8.8, the servicing of the second aperiodic request is completed, and τ_2 is resumed. A replenishment of 1.8 units of time is set for time = 12. The response time of the second aperiodic request is 2.8 units of time. At time = 11, the second replenishment of server execution time occurs, bringing the server's budget up to 0.2 units of time. At time = 11.8, τ_2 completes execution, τ_1 begins execution. At time = 12, the third replenishment of 1.8 units of time occurs, bringing the server's budget back to 2 units of time.

Figure 8 shows the behavior of the DXS algorithm. The server begins with its full execution budget of 2.0 unit. At time = 0, τ_1 begins execution and t_z is undefined. At time = 2, τ_1 completes execution and the first aperiodic request occurs and is serviced immediately since $d_S = t_z + T_S = 7$. At time = 3.8, the servicing of the first aperiodic request is completed, and τ_2 begins execution. Since there is no more request, the remaining 0.2 server execution time is discarded. A replenishment of 2 units of time is set for time = $2 + \frac{1.8}{2.0}5 = 6.5$ (denoted by arrow pointing from time = 0 on the task execution line to time = 6.5 on the server budget time line in Figure 8). The response time of the first aperiodic request is 1.8 units of time. At time = 6, the second aperiodic request occurs. It has to wait since there is no execution time budget left. At time = 6.5, the first replenishment of server execution time occurs, bringing the server's budget up to 2 units of time. t_z becomes defined and the second aperiodic request can now be executed since d_S is equal to 11.5. At time = 8.5, the servicing

of the second aperiodic request is completed, and τ_2 is resumed. A replenishment of 2 units of time is set for time = 11.5. The response time of the second aperiodic request is 2 units of time. At time = 11.5, the second replenishment of server execution time occurs, bringing the server's budget back up to 2 units of time. At time = 11.8, τ_2 completes execution, τ_1 begins execution.

7 Background Processing

The DDS, DSS, and DXS scheduling models can be combined with background processing. In such a hybrid model, the deferrable, sporadic, or exchange server can use either *foreground* or *background* processor time. The use of foreground time is scheduled according to the EDF algorithm; it is strictly budgeted, since while the server is executing in foreground it competes for time with hard-deadline tasks. The use of background time is unlimited, since then there are no competing hard-deadline requests. This can be expected to allow better response time for sporadic requests when the hard-deadline portion of the processing load is relatively light, so that a significant amount of background time is available.

Adding background service does not affect the worst-case schedulability of hard-deadline tasks. The resulting system can be modeled as if there were two servers. One of these is a pure background server. The other is a pure foreground (deferrable, sporadic or exchange) server. Any hard-deadline task can preempt the background server, except when it is in a critical section. Thus, the only contribution of the background server to the schedulability analysis is this blocking effect. Since this is no more blocking than could be caused by a pure foreground server, the analysis is unchanged.

8 Performance Studies

Simulations were conducted to compare the performance of the polling, DDS, DSS, and DXS algorithms for soft deadline aperiodic tasks. For comparability, these simulations were modeled after those reported for the DS and SS algorithms in [11].

8.1 Work Load

Three sets of ten periodic tasks were chosen, each with periods ranging from 5,400 to 120,000, having total processor utilizations of 40%, 69%, and 88%[1]. The specific periods and execution

[1]We chose specific task sets, rather than random sets of task periods and workloads, chiefly because we were not comfortable hypothesizing a suitable probability model; we are not aware of any work that characterizes the distribution of periods and execution times in the task sets of real-time applications. However, from our observation of the examples we have run, we believe these examples are "fair" in the sense of not provoking either extreme of performance.

	40% load		69% load		88% load	
i	T_i	C_i	T_i	C_i	T_i	C_i
1	5400	200	5400	600	5400	300
2	10800	600	14400	600	10800	1000
3	21600	1600	24000	500	21600	2800
4	27000	3000	43200	6400	30000	1400
5	36000	400	54000	3000	43200	7200
6	43200	1200	67500	4500	54000	11000
7	54000	1000	72000	7200	60000	3600
8	67500	1500	90000	3600	90000	6600
9	108000	1000	108000	2000	108000	2000
10	120000	4000	120000	10500	120000	4000

Table 1: Periodic task parameters

Server	40% load	69% load	88% load
Polling, DSS, DXS	3240	1674	648
DDS	3181	1622	623

Table 2: Deadline server sizes

times are shown in Table 1. The relative deadline of each task is equal to its period.

8.2 Short Server Period

In the first set of tests, the server period for each of the algorithms was chosen to be 5,400, i.e., the same as the shortest periodic task. The server size was chosen to be the largest for which schedulability of the periodic load could be guaranteed under the particular server algorithm. The actual server sizes for the experiments are shown in Table 2. Note that the server sizes given by the schedulability test for the DDS are smaller than for the DSS and DXS, though for these task sets the differences are not very large.

The aperiodic load was varied across the range of resource utilization unused by the periodic tasks. The interarrival times for the aperiodic tasks were generated with an exponential distribution. Three mean interarrival times were tested: 1,800, 3,600, and 5,400. The aperiodic service times were also modeled using an exponential distribution. Several mean service times were tested, for each mean interarrival time.

In the simulations, since the objective of an aperiodic server is to achieve low average response time for soft deadline aperiodic requests, we assumed an aperiodic request wins any priority ties with periodic tasks. *We did not simulate any blocking.*

In the results reported here, we did not allow the server to use any background time. We chose to present the results without background processing, primarily because we felt they

would be more representative of what would be achieved in a multi-server situation (with several servers running with different capacities and periods)[2].

The average response times of all the server scheduling models, for each of the combinations of mean periodic interarrival times and periodic loads that we tested are given in Tables 3-5. The first two columns are the periodic and aperiodic load. The number following "±" is the spread of the 99% confidence interval, expressed as a percentage of the average, rounded to the nearest whole percent. This was computed as $z_{0.995}s/\sqrt{n}$, where n is the number of aperiodic requests, s is the sample standard deviation of the response time, and $z_{0.995}$ is the 0.995th quantile of the standard normal distribution.

[2]We also found that the backgound processing tends to pay off most when the aperiodic load approaches or exceeds the server capacity, in which case the particular choice of foreground server scheduling algorithm no longer makes a significant difference in performance.

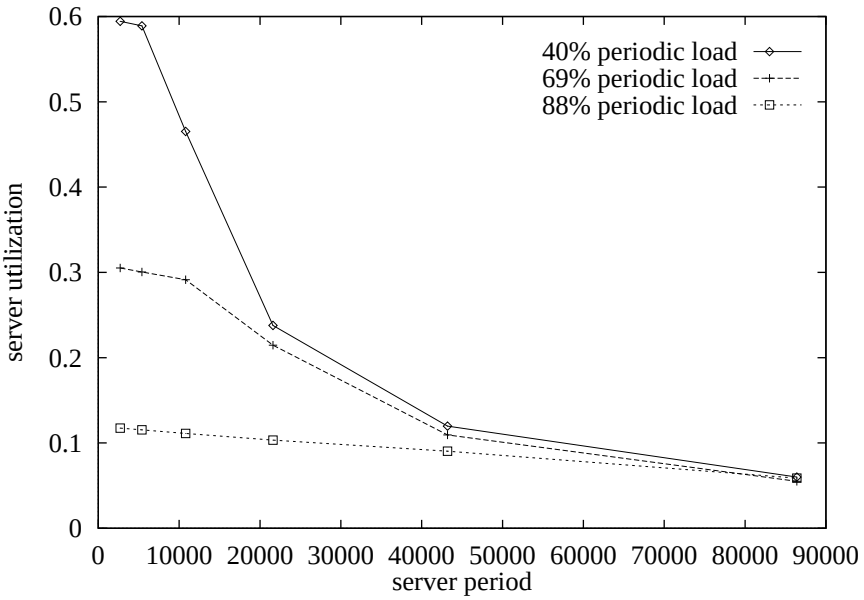

Figure 3: Maximum DDS server utilization related to server period

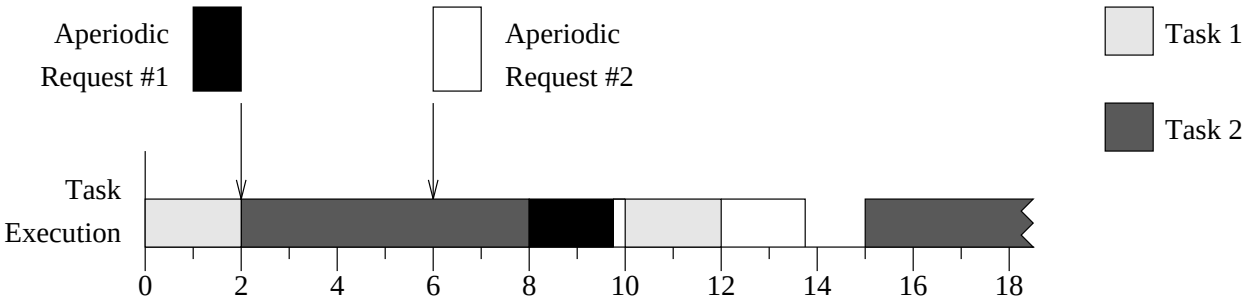

Figure 4: Background service example

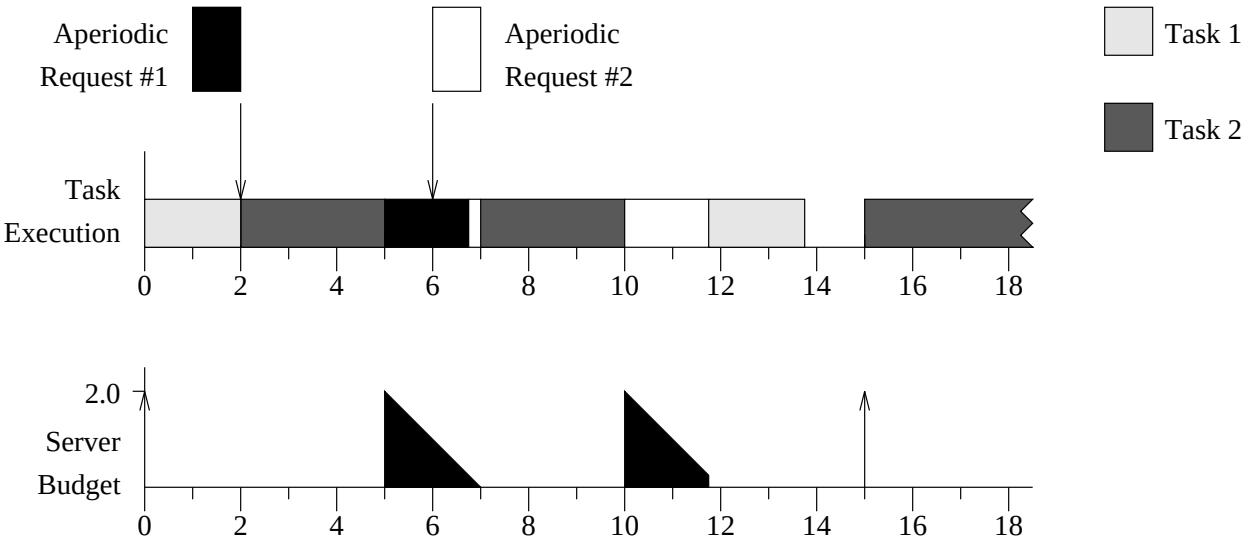

Figure 5: Polling service example

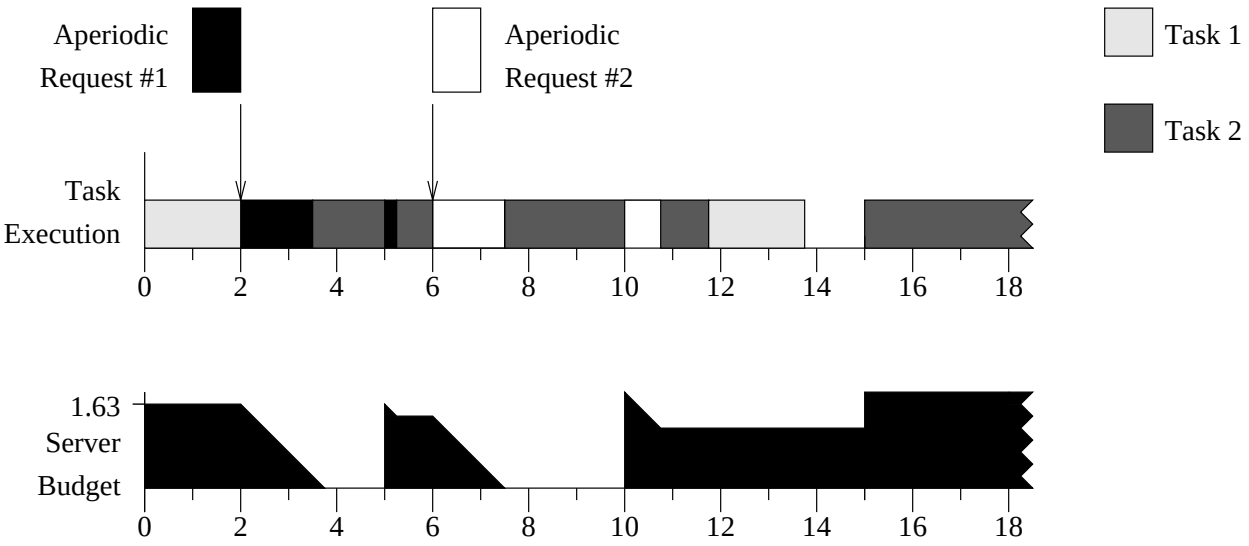

Figure 6: Deadline Deferrable Server example

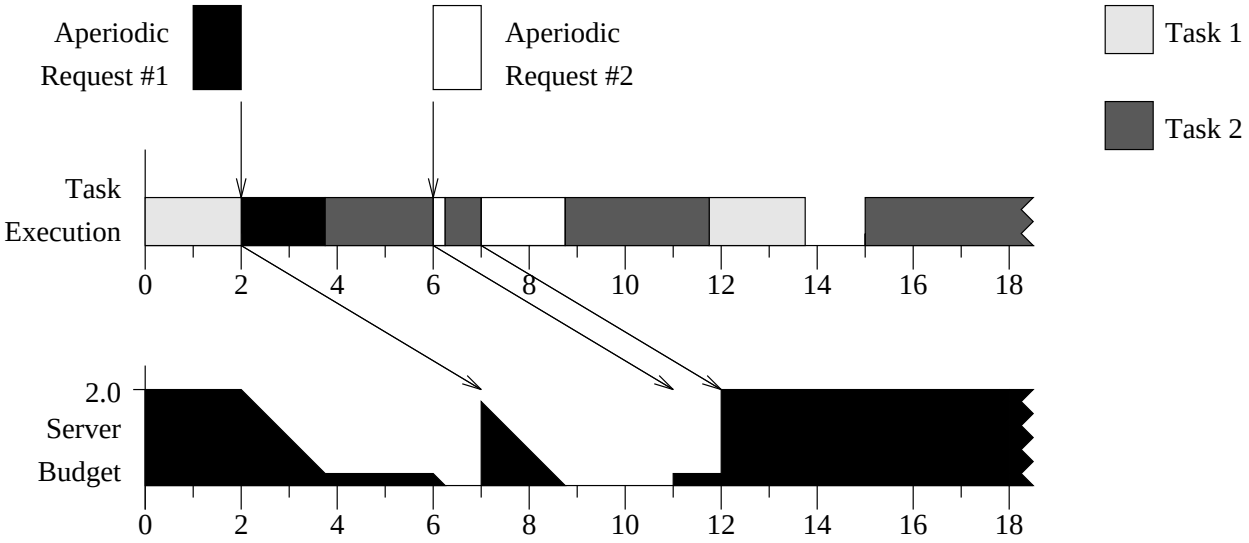

Figure 7: Deadline Sporadic Server example

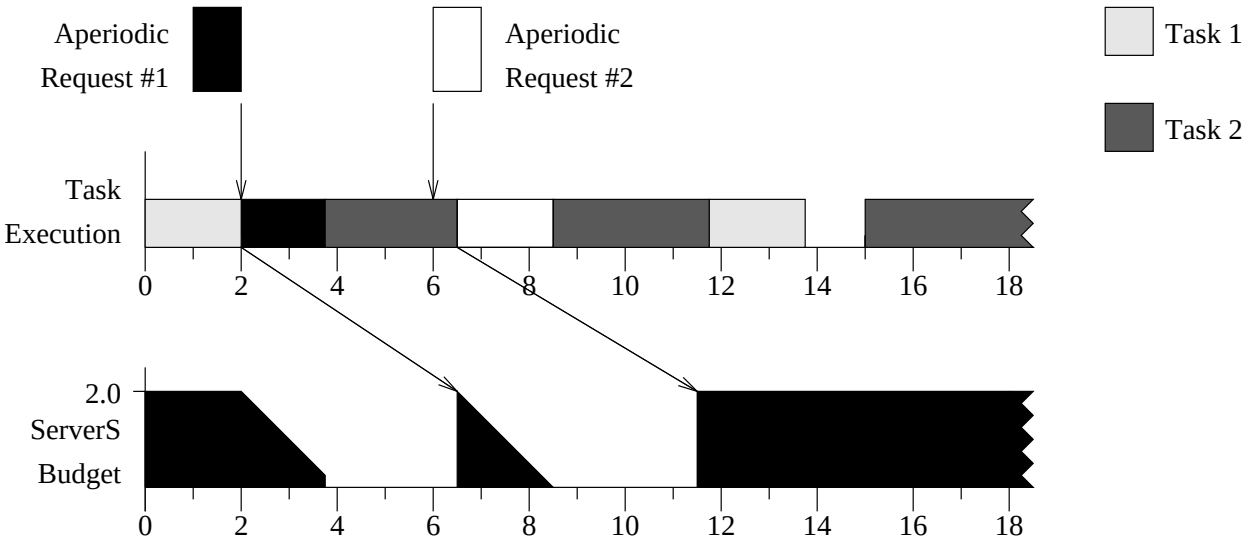

Figure 8: Deadline Exchange Server example

periodic	aperiodic	BGS	PLS	DDS	DSS	DXS
0.40	0.05	1161± 2.6%	2635± 0.8%	94 ± 3.3%	94 ± 3.3%	96± 3.3%
0.40	0.10	1410± 2.4%	2632± 0.9%	198 ± 3.6%	198 ± 3.6%	209± 3.5%
0.40	0.15	1700± 2.2%	2675± 1.0%	317 ± 4.4%	321 ± 4.3%	343± 3.9%
0.40	0.20	2055± 2.1%	2821± 1.2%	478 ± 5.0%	494 ± 4.7%	521± 4.3%
0.40	0.25	2498± 2.1%	3082± 1.4%	728 ± 5.2%	761 ± 4.7%	778± 4.5%
0.40	0.30	3069± 2.2%	3498± 1.7%	1115 ± 5.1%	1167 ± 4.6%	1160± 4.5%
0.40	0.35	3812± 2.2%	4084± 2.0%	1726 ± 4.8%	1769 ± 4.3%	1737± 4.4%
0.40	0.40	4847± 2.4%	5022± 2.2%	2677 ± 4.4%	2670 ± 4.1%	2617± 4.2%
0.40	0.45	6465± 2.5%	6510± 2.6%	4356 ± 4.1%	4199 ± 3.9%	4121± 4.0%
0.40	0.50	9600± 2.8%	9556± 2.9%	7843 ± 3.9%	7162 ± 3.8%	7059± 3.8%
0.40	0.55	17083± 3.0%	17063± 3.1%	16889 ± 3.8%	14496 ± 3.7%	14357± 3.7%
0.69	0.02	7639± 2.0%	2620± 0.9%	45 ± 3.3%	45 ± 3.3%	54± 3.4%
0.69	0.05	8310± 1.9%	2635± 0.8%	94 ± 3.7%	94 ± 4.1%	121± 3.6%
0.69	0.07	8936± 1.8%	2644± 0.9%	148 ± 5.3%	152 ± 6.0%	208± 4.3%
0.69	0.10	9554± 1.7%	2724± 1.1%	220 ± 6.9%	254 ± 7.0%	330± 5.1%
0.69	0.12	10327± 1.7%	2907± 1.3%	351 ± 7.6%	443 ± 6.8%	507± 5.5%
0.69	0.15	11329± 1.6%	3217± 1.6%	579 ± 7.4%	759 ± 6.0%	782± 5.6%
0.69	0.17	12651± 1.5%	3733± 1.9%	974 ± 6.7%	1252 ± 5.4%	1229± 5.3%
0.69	0.20	14486± 1.4%	4488± 2.2%	1644 ± 6.0%	1993 ± 4.8%	1912± 5.0%
0.69	0.22	16933± 1.3%	5746± 2.5%	2914 ± 5.3%	3226 ± 4.4%	3074± 4.6%
0.69	0.25	20802± 1.3%	8034± 2.8%	5589 ± 4.7%	5475 ± 4.1%	5245± 4.3%
0.69	0.27	27949± 1.5%	13099± 3.1%	12612 ± 4.1%	10518 ± 3.8%	10197± 4.0%
0.88	0.01	18269± 1.4%	2539± 0.9%	18 ± 3.3%	18 ± 3.3%	27± 4.1%
0.88	0.02	19317± 1.3%	2608± 0.9%	36 ± 4.5%	36 ± 6.0%	75± 4.4%
0.88	0.03	21047± 1.3%	2646± 0.9%	58 ± 8.6%	66± 11.6%	148± 5.1%
0.88	0.04	22885± 1.3%	2727± 1.0%	99± 11.9%	149± 11.3%	258± 5.9%
0.88	0.05	24613± 1.3%	2911± 1.3%	192± 12.2%	333 ± 9.0%	424± 6.4%
0.88	0.06	26840± 1.3%	3249± 1.6%	389± 10.4%	670 ± 7.0%	693± 6.3%
0.88	0.07	29569± 1.2%	3770± 2.0%	761 ± 8.8%	1206 ± 5.9%	1147± 5.9%
0.88	0.08	33091± 1.1%	4627± 2.3%	1494 ± 7.2%	2047 ± 5.0%	1881± 5.4%
0.88	0.09	38898± 1.1%	6114± 2.6%	3081 ± 5.8%	3510 ± 4.5%	3214± 4.9%
0.88	0.10	47190± 1.0%	9014± 2.9%	6836 ± 4.8%	6376 ± 4.1%	5950± 4.5%
0.88	0.11	63183± 0.9%	16114± 3.3%	19510 ± 3.9%	13518 ± 3.9%	12914± 4.1%

Table 3: EDF response times, mean aperiodic IAT 1805, after 54000000 time units.

periodic	aperiodic	BGS	PLS	DDS	DSS	DXS
0.40	0.05	1297± 3.3%	2751± 1.2%	189± 2.3%	189± 2.3%	194± 2.3%
0.40	0.10	1722± 2.8%	2892± 1.1%	408± 2.5%	409± 2.5%	424± 2.4%
0.40	0.15	2221± 2.5%	3175± 1.2%	708± 2.9%	717± 2.8%	728± 2.7%
0.40	0.20	2828± 2.3%	3615± 1.3%	1178± 3.0%	1179± 2.9%	1182± 2.8%
0.40	0.25	3592± 2.2%	4293± 1.5%	1868± 3.0%	1840± 2.9%	1826± 2.8%
0.40	0.30	4577± 2.1%	5230± 1.6%	2836± 2.8%	2757± 2.8%	2725± 2.8%
0.40	0.35	5926± 2.1%	6535± 1.7%	4224± 2.7%	4067± 2.6%	4026± 2.6%
0.40	0.40	7853± 2.0%	8418± 1.8%	6294± 2.5%	5967± 2.5%	5922± 2.5%
0.40	0.45	10950± 2.0%	11664± 1.9%	9863± 2.4%	9147± 2.4%	9089± 2.4%
0.40	0.50	17187± 2.1%	17923± 2.0%	17281± 2.3%	15434± 2.3%	15380± 2.3%
0.40	0.55	33152± 2.0%	34047± 2.0%	37971± 2.1%	31590± 2.2%	31543± 2.2%
0.69	0.02	7755± 2.8%	2704± 1.2%	92± 2.3%	92± 2.3%	105± 2.6%
0.69	0.05	8458± 2.6%	2766± 1.2%	192± 2.6%	200± 3.0%	242± 2.8%
0.69	0.07	9175± 2.5%	2950± 1.2%	348± 3.4%	394± 3.9%	441± 3.3%
0.69	0.10	9957± 2.4%	3303± 1.3%	628± 3.9%	754± 3.9%	770± 3.5%
0.69	0.12	10919± 2.2%	3850± 1.5%	1086± 3.8%	1300± 3.6%	1290± 3.4%
0.69	0.15	12171± 2.1%	4654± 1.7%	1800± 3.5%	2081± 3.3%	2027± 3.2%
0.69	0.17	13873± 1.9%	5808± 1.8%	2902± 3.3%	3210± 3.0%	3136± 3.0%
0.69	0.20	16137± 1.8%	7433± 1.9%	4626± 3.0%	4842± 2.8%	4735± 2.8%
0.69	0.22	19447± 1.7%	10043± 2.0%	7581± 2.8%	7455± 2.6%	7316± 2.6%
0.69	0.25	24976± 1.6%	14865± 2.0%	13664± 2.6%	12223± 2.5%	12075± 2.5%
0.69	0.27	36817± 1.6%	25691± 2.1%	29508± 2.3%	23092± 2.3%	22895± 2.3%
0.88	0.01	18621± 1.9%	2643± 1.2%	36± 2.4%	36± 2.4%	55± 4.1%
0.88	0.02	19531± 1.8%	2707± 1.2%	78± 3.7%	89± 5.5%	153± 4.3%
0.88	0.03	21091± 1.8%	2863± 1.2%	168± 5.6%	246± 6.1%	318± 4.5%
0.88	0.04	22793± 1.8%	3209± 1.4%	380± 5.6%	595± 5.1%	628± 4.4%
0.88	0.05	24708± 1.8%	3795± 1.6%	787± 4.9%	1168± 4.2%	1132± 4.0%
0.88	0.06	27154± 1.8%	4639± 1.7%	1492± 4.2%	2006± 3.6%	1897± 3.6%
0.88	0.07	30116± 1.7%	5880± 1.9%	2668± 3.7%	3253± 3.2%	3079± 3.2%
0.88	0.08	34447± 1.6%	7705± 2.0%	4632± 3.3%	5098± 2.9%	4876± 3.0%
0.88	0.09	40892± 1.5%	10858± 2.0%	8397± 2.9%	8213± 2.6%	7923± 2.7%
0.88	0.10	51410± 1.4%	17075± 2.1%	17078± 2.6%	14411± 2.5%	14094± 2.5%
0.88	0.11	71360± 1.3%	32742± 2.1%	46113± 2.1%	30341± 2.3%	29966± 2.3%

Table 4: EDF response times, mean aperiodic IAT 3605, after 54000000 time units.

periodic	aperiodic	BGS	PLS	DDS	DSS	DXS
0.40	0.05	1467± 3.7%	2854± 1.4%	283± 2.7%	283± 2.7%	291± 2.7%
0.40	0.10	2053± 3.1%	3152± 1.4%	639± 3.2%	640± 3.1%	651± 3.0%
0.40	0.15	2771± 2.8%	3688± 1.5%	1189± 3.5%	1184± 3.4%	1186± 3.3%
0.40	0.20	3637± 2.6%	4453± 1.7%	1980± 3.4%	1943± 3.3%	1934± 3.3%
0.40	0.25	4782± 2.5%	5531± 1.9%	3113± 3.3%	2994± 3.2%	2973± 3.2%
0.40	0.30	6264± 2.5%	6960± 2.0%	4657± 3.1%	4428± 3.1%	4396± 3.1%
0.40	0.35	8277± 2.4%	8972± 2.1%	6829± 2.9%	6443± 2.9%	6410± 2.9%
0.40	0.40	11164± 2.4%	11837± 2.2%	9947± 2.8%	9323± 2.8%	9286± 2.8%
0.40	0.45	15958± 2.5%	16762± 2.3%	15426± 2.7%	14243± 2.7%	14206± 2.8%
0.40	0.50	25666± 2.5%	26341± 2.4%	26838± 2.7%	23838± 2.7%	23810± 2.7%
0.40	0.55	48797± 2.4%	50138± 2.4%	57364± 2.4%	47673± 2.5%	47675± 2.5%
0.69	0.02	7984± 3.4%	2758± 1.4%	138± 2.7%	138± 2.7%	156± 2.9%
0.69	0.05	8844± 3.2%	2925± 1.4%	311± 3.7%	336± 4.2%	380± 3.8%
0.69	0.07	9670± 3.0%	3325± 1.6%	644± 4.5%	738± 4.6%	757± 4.3%
0.69	0.10	10665± 2.8%	3970± 1.8%	1191± 4.4%	1371± 4.2%	1365± 4.1%
0.69	0.12	12001± 2.6%	4904± 2.0%	2032± 4.1%	2294± 3.8%	2259± 3.8%
0.69	0.15	13757± 2.4%	6208± 2.1%	3257± 3.8%	3558± 3.5%	3496± 3.5%
0.69	0.17	16010± 2.3%	7996± 2.2%	5082± 3.5%	5351± 3.2%	5277± 3.3%
0.69	0.20	18873± 2.2%	10511± 2.3%	7783± 3.2%	7840± 3.0%	7749± 3.1%
0.69	0.22	23034± 2.1%	14566± 2.4%	12522± 3.0%	11881± 2.9%	11812± 2.9%
0.69	0.25	31159± 2.0%	21778± 2.4%	21909± 2.8%	19198± 2.8%	19108± 2.8%
0.69	0.27	49076± 2.1%	37982± 2.5%	44986± 2.6%	35438± 2.7%	35329± 2.7%
0.88	0.01	19131± 2.3%	2682± 1.5%	54± 2.8%	54± 2.7%	82± 4.7%
0.88	0.02	20083± 2.2%	2812± 1.5%	142± 6.3%	179± 7.4%	246± 5.7%
0.88	0.03	21730± 2.2%	3195± 1.7%	381± 6.6%	557± 6.3%	581± 5.6%
0.88	0.04	23724± 2.2%	3853± 1.9%	850± 5.8%	1201± 5.0%	1175± 4.9%
0.88	0.05	26097± 2.2%	4842± 2.1%	1684± 4.9%	2170± 4.3%	2090± 4.3%
0.88	0.06	28939± 2.1%	6232± 2.2%	2944± 4.3%	3544± 3.7%	3403± 3.8%
0.88	0.07	32220± 2.0%	8195± 2.3%	4986± 3.8%	5511± 3.4%	5342± 3.4%
0.88	0.08	37152± 1.9%	11010± 2.4%	8182± 3.4%	8330± 3.1%	8137± 3.2%
0.88	0.09	44229± 1.8%	15828± 2.5%	14176± 3.1%	13165± 2.9%	12963± 3.0%
0.88	0.10	56085± 1.7%	25301± 2.5%	27490± 2.9%	22677± 2.8%	22454± 2.8%
0.88	0.11	83081± 1.6%	48987± 2.4%	67649± 2.3%	46317± 2.6%	46052± 2.6%

Table 5: EDF response times, mean aperiodic IAT 5395, after 54000000 time units.

It can be seen that the average response times grow very quickly as the mean processor load approaches 100%. For these load conditions, the variation in response time is also rather wide, as indicated by the large confidence intervals. In this range, due to the random aperiodic arrivals, the local average processor load often exceeds 100% for significant blocks of time. For all practical purposes, the processor is overloaded.

The same data are presented graphically in Figures 9-14. In all these figures, the average response time of each algorithm is plotted as a function of the aperiodic load. Lower values indicate better performance. The smooth lines indicate the predicted performance based on queueing models.

The lower smooth line, labeled "M/M/1" corresponds to the function

$$\mathcal{L}(\rho) = \frac{\rho}{\lambda(1-\rho)}$$

where ρ is the mean aperiodic load (the horizontal axis of the graph), and $\frac{1}{\lambda}$ is the mean aperiodic interarrival time. This is the expected response time for a pure M/M/1 model[3], in which there is no periodic load, i.e. 100% of the processor capacity is available for the aperiodic tasks.

The upper smooth line, labeled "M/M/c", for $c = 0.6$, 0.31, and 0.12, corresponds to the function

$$\mathcal{U}(\rho) = \frac{\rho}{\lambda(c-\rho)}$$

where $c = \frac{C_S}{T_S}$. This is the expected response time for a scaled M/M/1 model, in which the aperiodic tasks are served by a virtual processor whose speed has been scaled to c, the fraction of the processor that is not used by hard-deadline periodic tasks. For example, if the hard-deadline periodic tasks utilize 40% of the processor, the capacity of the virtual processor serving aperiodic tasks is 60%. This is the limit of the polling server performance, as the server period approaches zero.

As can be seen in all the graphs, the DDS, DSS, and DXS algorithms all provide much better performance than polling. They provide the greatest improvement for lower aperiodic loads, where the performances of all three algorithms are essentially indistiguishable. As the aperiodic load goes up, some differences between the algorithms appear. The DXS does slightly better than the DSS for moderate and high aperiodic loads. The DDS does better than the other two, for low and moderate aperiodic loads. The latter is somewhat surprising, since the maximum server size that can be supported for the DDS is smaller. However, the simulations indicate that this disadvantage is not noticed until the aperiodic load gets large. For very high aperiodic loads, the DDS does worse than the others, but this disadvantage may not have much practical importance, since under these circumstances no scheduling method

can give short average response times. This effect can be seen clearly in the graphs, where the behaviors of all servers converge to the scaled (M/M/c) queueing model for large values of ρ.

Apparently, the M/M/c and M/M/1 models provide an envelope that can be useful for predicting the performance of all the deadline aperiodic servers. Looking more closely, we see some asymptotic relationships. For the polling server, the average queuing time runs asymptotic to $\mathcal{U}(\rho)$ (upper broken line) as the aperiodic load approaches the server capacity, but for light aperiodic loads it is dominated by the effect of waiting for the next polling point, which is approximately one half the server period. For the DDS, DSS, and DXS, the average response times fall between $\mathcal{L}(\rho)$ and $\mathcal{U}(\rho)$. They are asymptotic to $\mathcal{L}(\rho)$ as the aperiodic load approaches zero, and asymptotic to $\mathcal{U}(\rho)$ as the aperiodic load approaches the server capacity, c. Intuitively, this makes sense. When the aperiodic load is light, it is very likely that an arriving aperiodic request will find the server has enough budget for the request to be served immediately — as it would if there were no competing periodic load. Thus, the behavior is like the unscaled M/M/1 model. In contrast, when the load is heavy, it is likely that an arriving request will not be able to be served to completion within the available server time budget, but will need to yield to the periodic tasks while it waits for replenishment. Thus, performance is close to the scaled (M/M/c) model.

The performance of the three algorithms in these tests is very close, especially if one takes into account the confidence intervals of the simulation. Given the similar peformances, the DDS could be considered the winner, since it is the simplest to implement.

8.3 Longer Server Periods

The first set of tests were based on the presumption that the server period is always the same as the shortest period of the the hard-deadline periodic tasks. However, for a fixed server utilization, increasing the server budget (and lengthening the server period proportionally, to keep the utilization constant) should increase the probability that an arriving aperiodic request can be served to completion. That is, the range of aperiodic loads over which the behavior is close to the M/M/1 model should be larger. To test this hypothesis, the experiments described above were repeated, for a range of different server periods. A sample of the results of these simulations for the DDS, DSS, and DXS is shown in Figures 15-17, for a mean interarrival time of 3600 with 69% periodic load.

The data for the DDS stops at period 10800. Above that, it overloads, since server size must be made very small to guarantee schedulability of the hard deadline tasks. Up to that point, the larger the server period, the shorter the average response time.

In contrast to the DDS, the DSS and DXS do not overload as the server period increases. However, there is a sudden reversal of behavior as the period gets sufficiently large. For the DSS, the reversal occurs between server periods 43200 and 86400. For the DXS, it starts a little earlier, between periods 21600 and 43200. With other combinations of periodic and aperiodic load, these jumps occur for different server periods.

It seems there are several conflicting effects at work:

(1) A longer server period means a higher probability that an arriving request can be served to completion without waiting for a server replenishment.

(2) A longer server period means the server generally has lower priority. For example, if the processor is idle when an aperiodic request arrives, the server's deadline is the arrival time of the aperiodic request plus the server period. A longer server period means a later deadline, which means a lower priority. Of course, this does not matter immediately if the processor was idle, but it means the server may be preempted by any period task with a shorter period.

(3) The server's priority is effectively raised if there has been a recent history of near-deadline activity. (See cases 3 and 4 of the definition of t_z, the time at which the server deadline becomes active.) If a request arrives while the processor is busy executing a (periodic) task with shorter deadline, the deadline of the aperiodic request will be measured (ignoring details related to the times of replenishments) from *before* the arrival time of the request — namely, from the earliest that would not have caused the server to preempt the processing that has gone on up to the time of the request. In the extreme, if the processor has recently been busy executing a task that is near its deadline, the server may be able to preempt that task.

The interactions of these (and other) effects, which depend on random events as well as the periods and sizes of the server and periodic tasks, seem difficult to analyze in general. However, it seems the reversal of the trend toward shorter average queuing times for sufficiently large server periods may be at least partially explained by effect (3).

Consider the case shown in Figure 16. In the 69% periodic task set, for a server period of 43200, only tasks 1-3 can preempt the server. That is, these are the only tasks that can arrive after the server has started a request and preempt the server before it uses up its budget. Of these, the longest execution time is 600, so this is the worst-case delay that the server can experience due to a single such preemption. However, for a server with period 86400, tasks 4-7 can also preempt, and these have execution times in the range 3000-7200. Clearly, both the number of tasks that can preempt the server, and the duration of the preemption, are

Server	40% load	69% load	88% load
Polling, SS	3160	1109	125
DS	2600	1085	117

Table 6: Fixed-priority server sizes

much larger. This large increase in effect (3) may be a cause of the large jump in average response time between periods 43200 and 86400 for the DSS and DXS in this example.

This does not yet explain the difference between the DSS and the DXS, which shows some degradation already for server period 43200. Periodic task 4 has execution time 6400. This is the first task whose execution time is long enough that there is a high probability of more than one aperiodic request occurring during its execution. The expected number of aperiodic arrivals during each execution of task 4 is 6400/3600. With the DSS, the residual server budget is likely to be high enough that all such requests can be served at the server priority, and so the server will be able to preempt task 4. In contrast, with the DXS, the server will be able to preempt the first time, but the next request will need to wait until the server is replenished, since between requests the server will have given up its budget. This is a price the DXS pays, compared to the DSS, for not allowing more than one replenishment to be pending at the same time.

9 Comparison with Rate Monotone Scheduling

For comparison, we repeated the simulations using fixed-priority versions of the aperiodic server algorithms under the same load conditions, and with a rate monotone priority assignment for the periodic tasks. The server sizes, shown in Table 6, were necessarily smaller. In this table, "SS" denotes the sporadic server, and "DS" denotes the deferrable server.

The same periodic task sets were used as in the previous simulations. While these task sets are not completely harmonic, they were considered fair tests for rate rate-monotonic scheduling, since a processor utilization of 88% could be achieved, as compared to the worst-case Liu and Layland bound of 72% and the best case harmonic bound of 100%.

In each case, the largest safe server size was determined by bisection, using a schedulabilty test similar to that described in [6]. The test computes the completion time W_k of periodic task τ_k, for the critical instant when all tasks are requested simultaneously. Task k is schedulable iff its completion time is before its deadline, i.e. $W_k \leq T_k$. W_k is computed iteratively, as the least fixed-point of the recurrence

$$
\begin{aligned}
W_0 &= 0 \\
W_k(n+1) &= C_k + \sum_{i=1}^{k-1} \left\lceil \frac{W_k(n)}{T_i} \right\rceil C_i + \text{Server\_Time}(W_k(n))
\end{aligned}
$$

The term Server_Time(Δ) denotes the maximum server execution time during a time-window of size Δ, which varies from one server scheduling method to another. The worst-case server execution time for the sporadic server is the same as a similar periodic task, i.e.

$$\text{Server\_Time}_{SS}(\Delta) = \left(\left\lceil \frac{\Delta}{T_S} \right\rceil \right) C_S.$$

The worst-case server execution time for the deferrable server, which is derived in [7], is

$$\text{Server\_Time}_{DS}(\Delta) = \left(1 + \left\lceil \frac{\Delta - C_S}{T_S} \right\rceil \right) C_S.$$

A sample of the results of the fixed-priority experiments for the deferrable server is shown in Figures 18-20. Each graph shows the results for 69% periodic load. The average response times of the DDS and DS are shown as a function of the aperiodic load, for three average interarrival times.

As expected, deadline scheduling allows higher utilization than rate monotone scheduling, and consequently allows larger server sizes. This results in much better average response time. It is especially noticeable for the simulations with higher processor utilization, where the rate monotone servers simply cannot handle the load.

The results of our deadline *versus* rate monotone comparision experiments for the other forms of servers and for other mean interarrival times are not given here, for brevity and because they were not surprising. The gross advantage of deadline scheduling, due to larger server sizes, was comparable across all the experiments. The other thing that was apparent is that the differences in performance between the deferrable and sporadic servers for fixed-priority scheduling were greater than with deadline scheduling, as would be expected from the larger gaps between the server sizes shown in Table 2 *versus* Table 6.

10 Conclusion

We have shown how the deferrable server and sporadic server algorithms can be adapted to be used with EDF scheduling. The resulting algorithms—the deadline deferrable server algorithm, the deadline sporadic server algorithm, and the deadline exchange server algorithm—can improve average response times for soft deadline aperiodic tasks, while still guaranteeing the deadlines of hard deadline periodic and aperiodic tasks.

The reasons for the improvement of average response times for soft deadline aperiodic tasks under these algorithms, compared with background processing and polling, can be explained intuitively. Compared with background processing, where aperiodic requests are only executed whenever the processor would otherwise be idle, the improvement comes from the high-priority processing time that is allocated to service aperiodic requests. If the load of the periodic task set is high, then utilization left for background service is low, and background service opportunities are relatively infrequent. Compared with polling, the improvement comes from preserving the high-priority time allocated for servicing aperiodic requests if, upon the invocation of the server task, no aperiodic requests are pending. In contrast, with polling, if an aperiodic request occurs shortly after the polling task has suspended, then the aperiodic request must wait until the beginning of the next polling task period (or until the processor is idle) before being serviced.

The deadline deferrable server, deadline sporadic server, and deadline exchange server algorithms differ in their effect upon the schedulability bound for periodic tasks. The DSS pays a significant schedulability penalty (in terms of a lower schedulable utilization bound). In contrast, under both the DSS and DXS algorithms, because of the execution time replenishment method, aperiodic requests will be forced to limit their use of high-priority time to what would be used by periodic requests. This allows the capacity of the aperiodic server to be larger before it starts causing other tasks to miss hard deadlines, but the average response times will be slightly worse for servers of identical size.

The results of the simulation studies show that the relationship of the performance of the DDS compared to the DXS and DSS depends on the server period and the work load. All three algorithms yield similar peformance, if the server period is the same as the shortest period of the periodic tasks. For moderate workloads, the DDS performs slightly better than the other two algorithms, despite the slightly smaller server size. However, the situation changes for longer server periods, since the DDS requires a reduction in server capacity, while the DXS and DSS do not. By lengthening the server period, the DXS and DSS can achieve significantly better performance than the DDS. The DXS and DSS have very similar performance under all circumstances. Since the DXS is simpler to implement, the DXS may be the algorithm of choice, over all.

The principles governing the choice of server size and period are similar for all three forms of deadline aperiodic server. Given a class of aperiodic tasks for which the arrival and execution time distributions are known, the best server size depends on the nature of the deadline(s). The server capacity, $\frac{C_S}{T_S}$ must be greater than the expected aperiodic load. If the tasks have a hard deadline, an upper bound on the aperiodic tasks' execution times, $C_{\max}$, and a lower bound on the inter-arrival time, $T_{\min}$, must be known. The server period, T_S,

must be less than or equal to the hard deadline, and the server size must be large enough that $\frac{C_S}{T_S} \geq \frac{C_{\max}}{T_{\min}}$. If the tasks have a soft deadline, the server size and period must be chosen so that the average response time satisfies this soft deadline.

Generally, good average response time can be obtained by letting the server period be the same as the shortest period of the periodic tasks. Under this rule, the simulation studies show that the M/M/c queueing model can be used to obtain an upper bound for the average response time for all three of the aperiodic servers.

However, with the DDS, DSS, and DXS, better average response time can be achieved by using a longer longer server period, up to some limit determined by the particular set of competing period tasks. Where there are different kinds of aperiodic tasks, with different requirements, several servers may be used, with different periods and sizes.

In summary, we have addressed the problem of improving average response times for soft deadline aperiodic tasks, by demonstrating that techniques developed for handling aperiodic tasks with fixed priority rate monotone scheduling can be extended to EDF scheduling. We have shown that schedulability analysis can be done for systems involving an aperiodic server, in combination with hard deadline periodic and aperiodic tasks and blocking due to resource locking. This adds to the evidence in support of EDF scheduling as an alternative to fixed priority scheduling.

Acknowledgements

The authors are grateful to the referees for pointing out several places where the presentation needed clarification, and to Texas Instruments for providing partial support for this research.

References

[1] T.P. Baker, "Stack-Based Scheduling of Realtime Processes", *The Real-Time Systems Journal 3,1* (March 1991) 67-100.

[2] Chen, M.I., and K.J. Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems", Technical Report UIUCDCS-R-89-1511, Department of Computer Science, University of Illinois at Urbana-Champaign (1989).

[3] Coffman, E.G. Jr. and P.J. Denning, *Operating Systems Theory*, Prentice-Hall, New Jersey (1973) 152-165.

[4] Ghazalie, T., "Improving Response Time of Aperiodic Tasks with Deadline Scheduling", thesis for Master of Science, Department of Computer Science, Florida State University (May 1991).

[5] "Scheduling Sporadic Tasks with Shared Resources in Hard-Real-Time Systems", *Proceedings of the IEEE Real-Time Systems Symposium* (1992) 89-99.

[6] Joseph and Pandya, "Finding Response Times in a Real-Time System", *BCS Computing Journal, 29, 5* (October 1986) 390-395.

[7] Lehoczky, J.L., L. Sha, and J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *Proceedings of the IEEE Real-Time Systems Symposium* (1987) 261-270.

[8] Leung, J.Y.T., and M.L. Merrill, "A Note on Preemptive Scheduling of Periodic, Real-Time Tasks", *Information Processing Letters, 11,3* (November 1989)115-118.

[9] Liu, C.L., and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the Association for Computing Machinery, 20,1* (January 1973) 46-61.

[10] Sha, L., R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Transactions on Computers, 39,9* (September 1990) 1175-1185.

[11] Sprunt, B., L. Sha, and J.P. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems" *The Journal of Real-Time Systems, 1,1* (June 1989) 27-60.

[12] Strosnider, J.K, J.L. Lehoczky, and L. Sha, "The Deferrable Server Algorithm For Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *IEEE Transactions on Computers* (to appear).

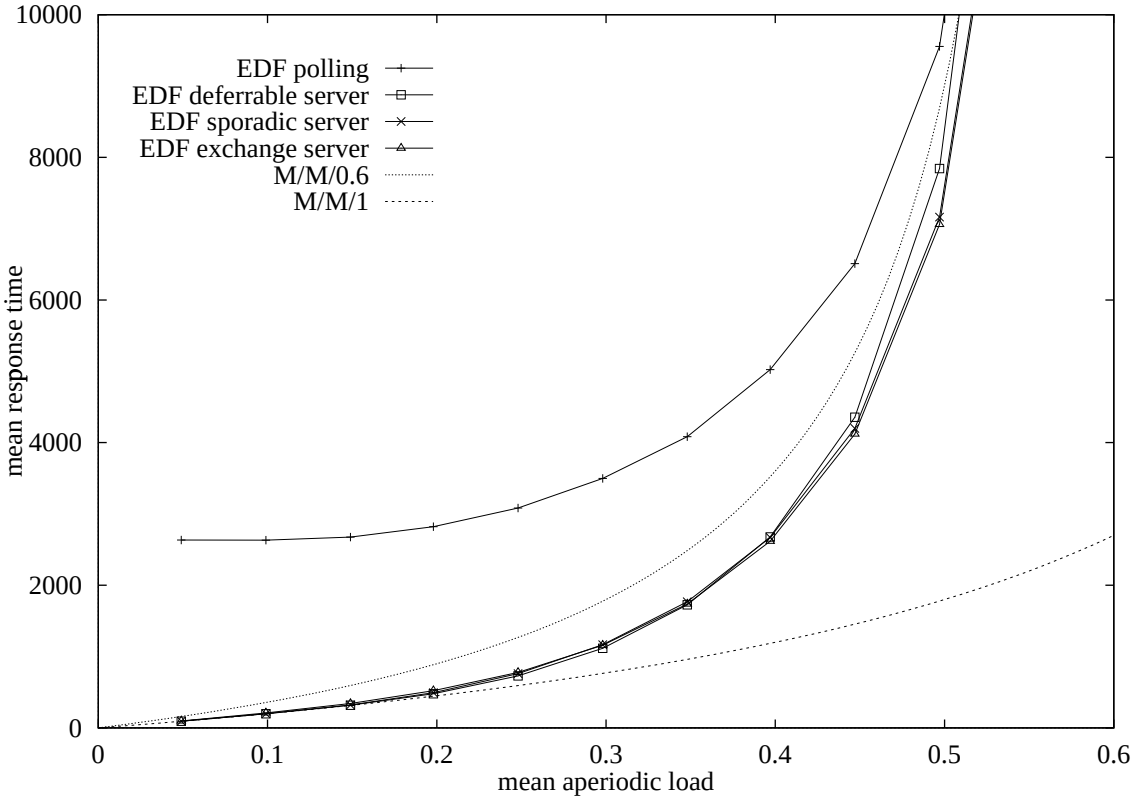

Figure 9: Mean aperiodic interarrival time 1800, 40% periodic load

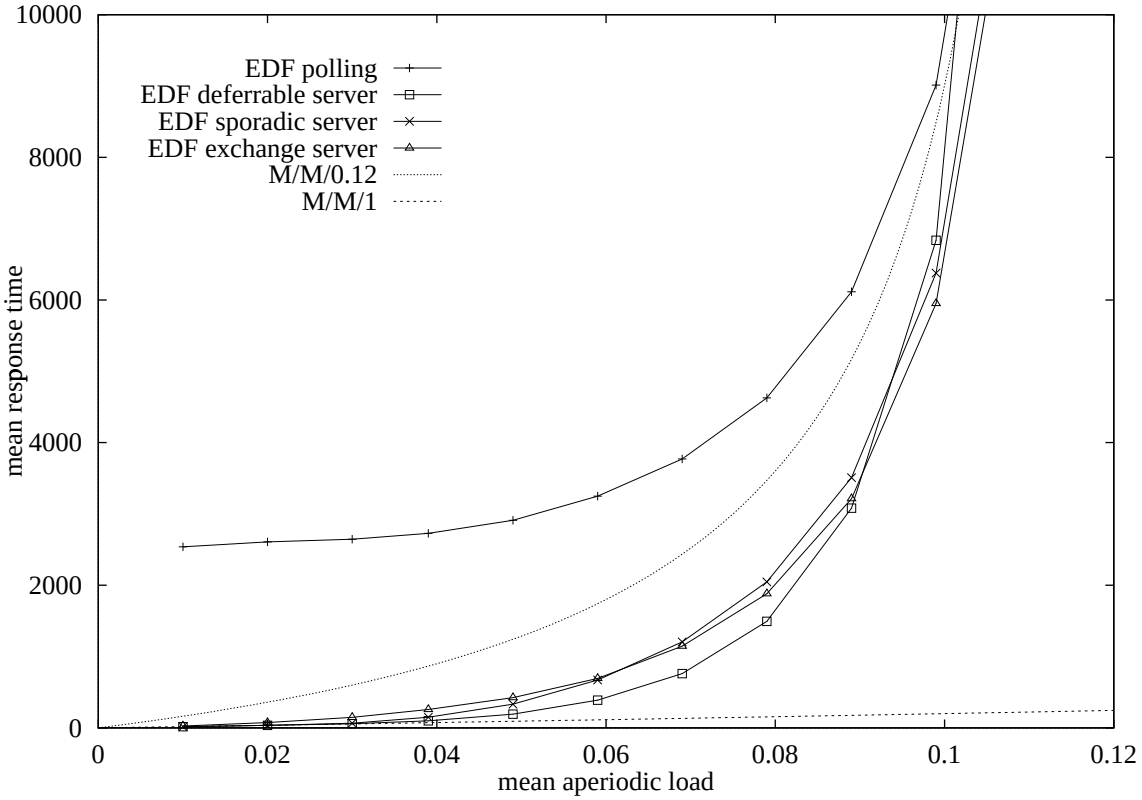

Figure 10: Mean aperiodic interarrival time 5400, 40% periodic load

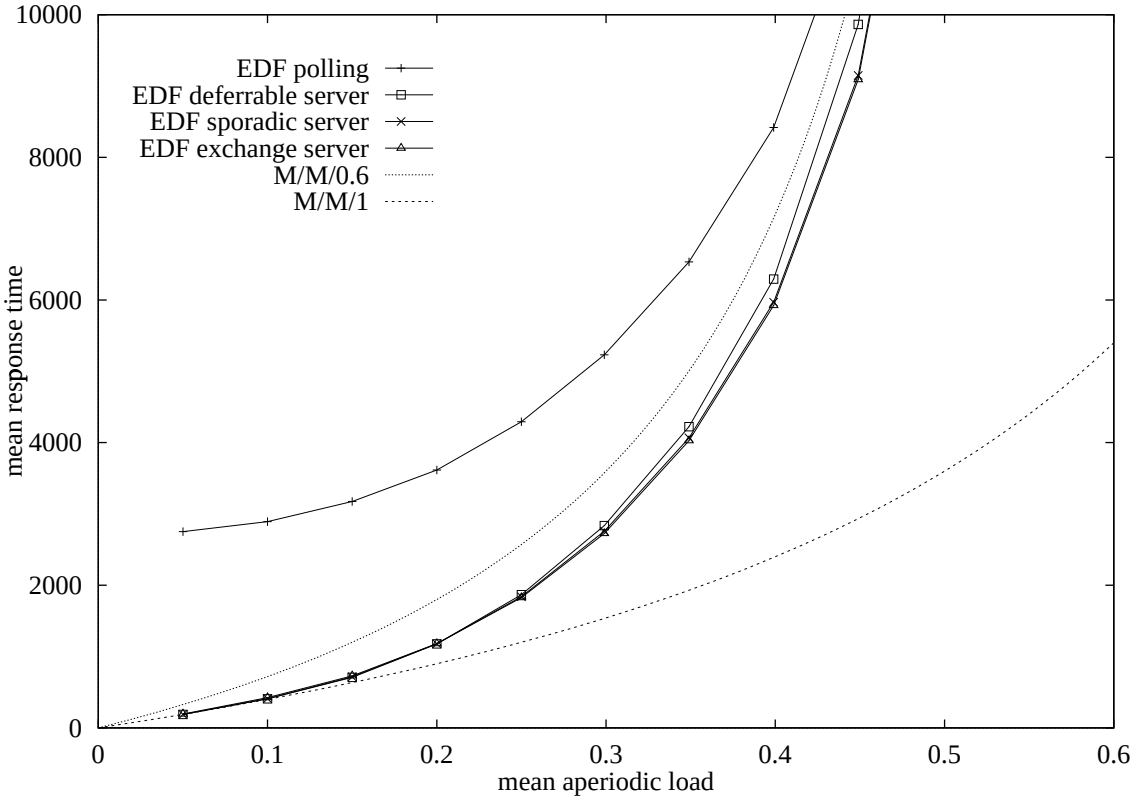

Figure 11: Mean aperiodic interarrival time 1800, 69% periodic load

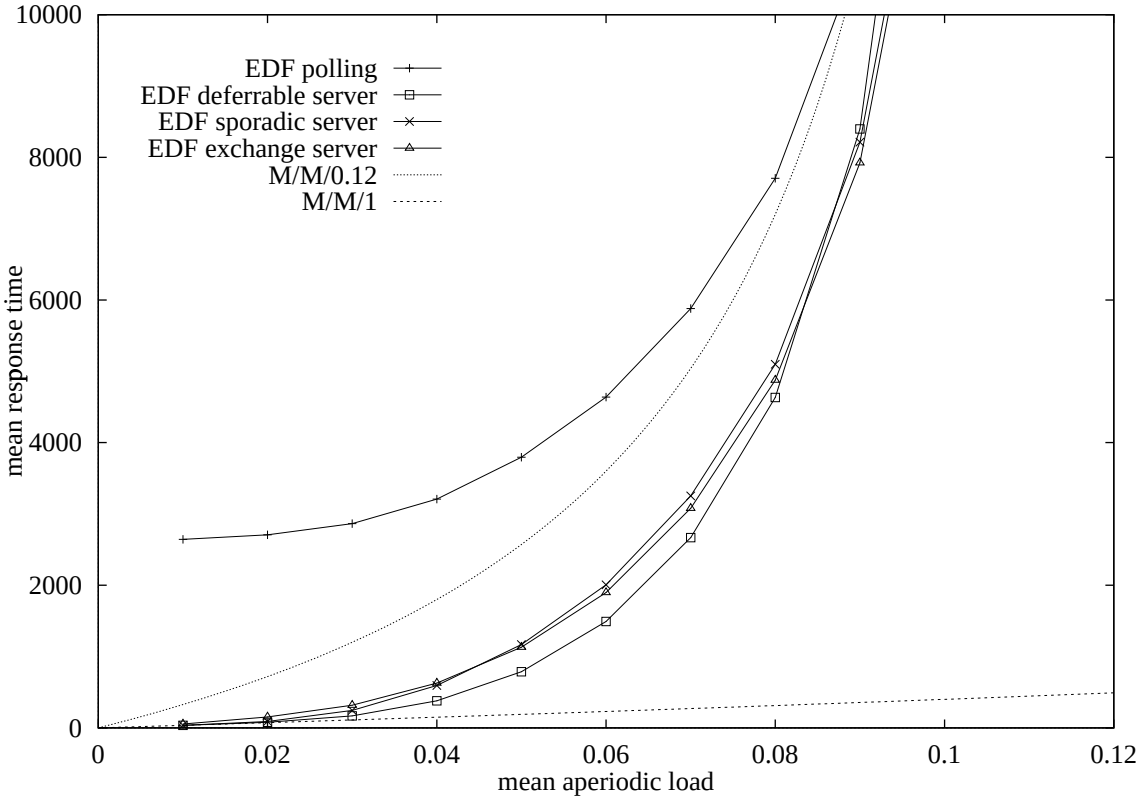

Figure 12: Mean aperiodic interarrival time 5400, 69% periodic load

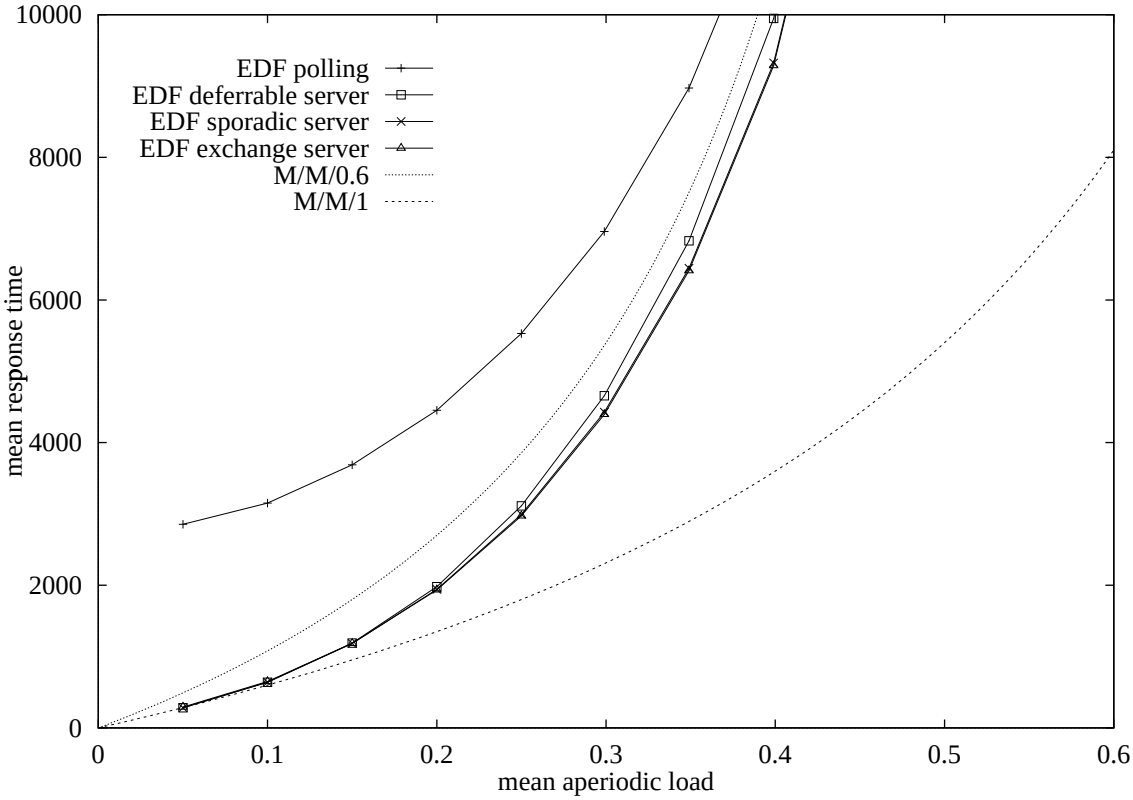

Figure 13: Mean aperiodic interarrival time 1800, 88% periodic load

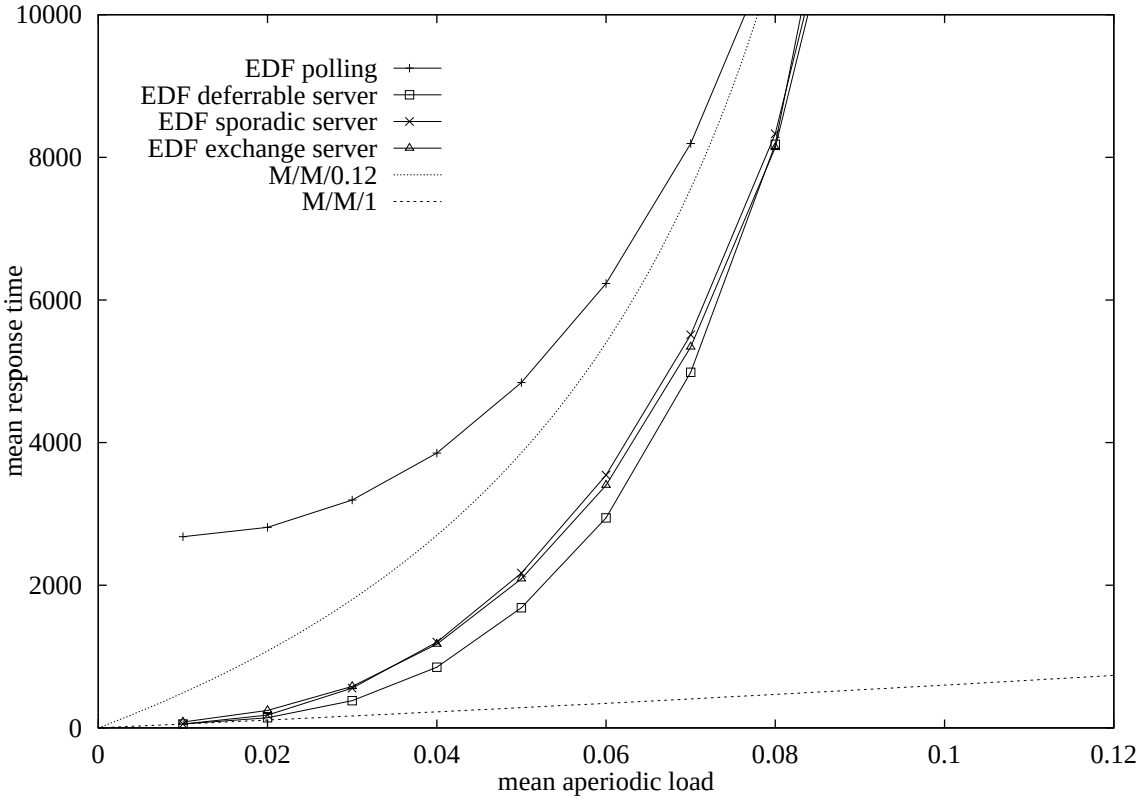

Figure 14: Mean aperiodic interarrival time 5400, 88% periodic load

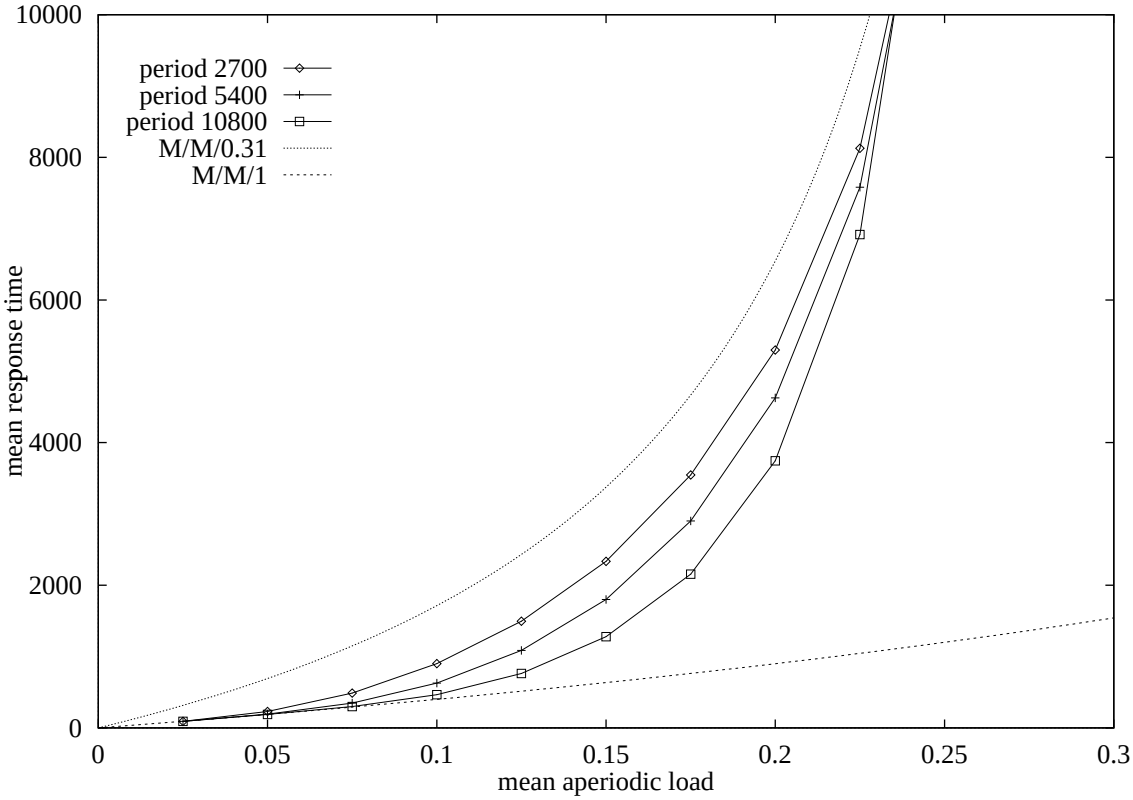

Figure 15: DDS response time versus period, IAT 3600, 69% periodic load.

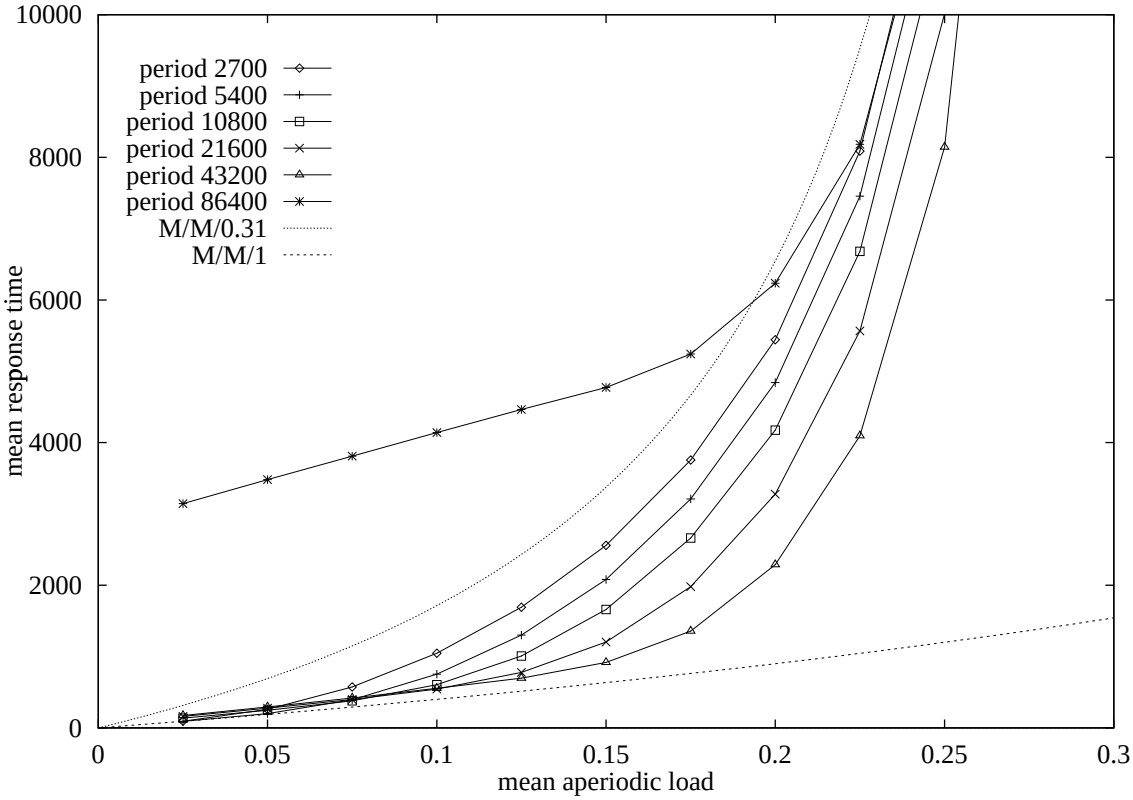

Figure 16: DSS response time versus period, IAT 3600, 69% periodic load.

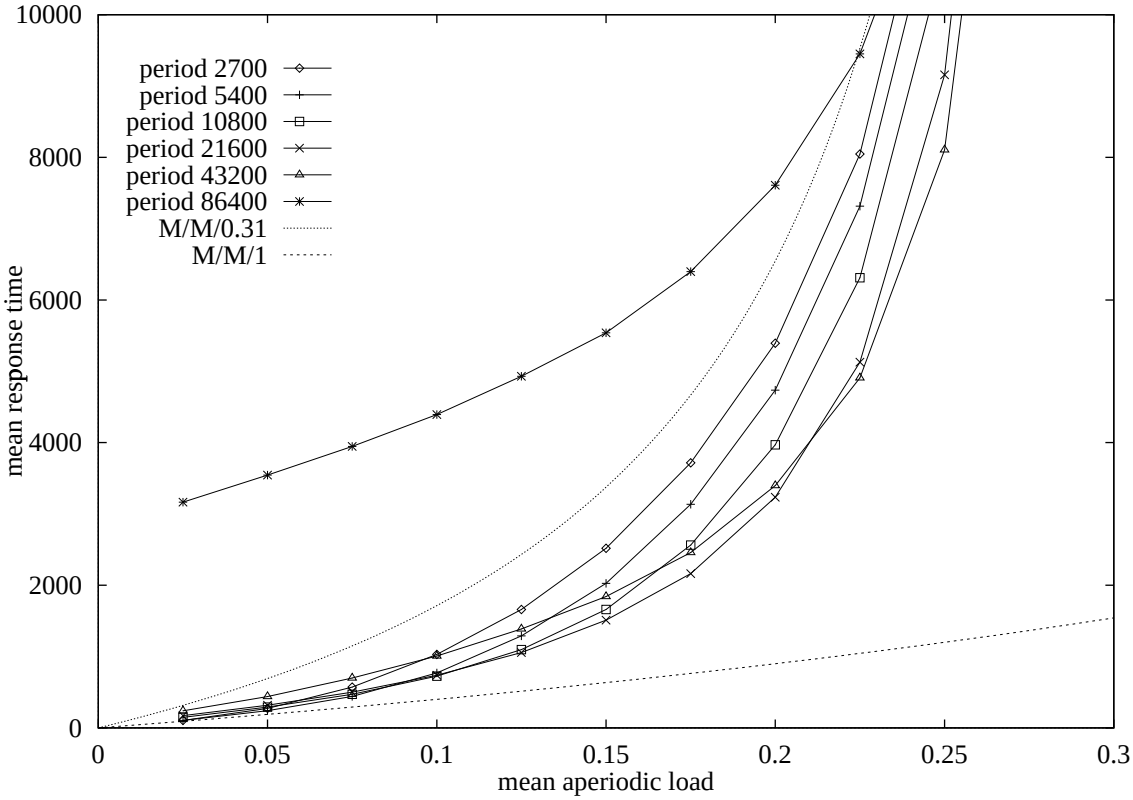

Figure 17: DXS response time versus period, IAT 3600, 69% periodic load.

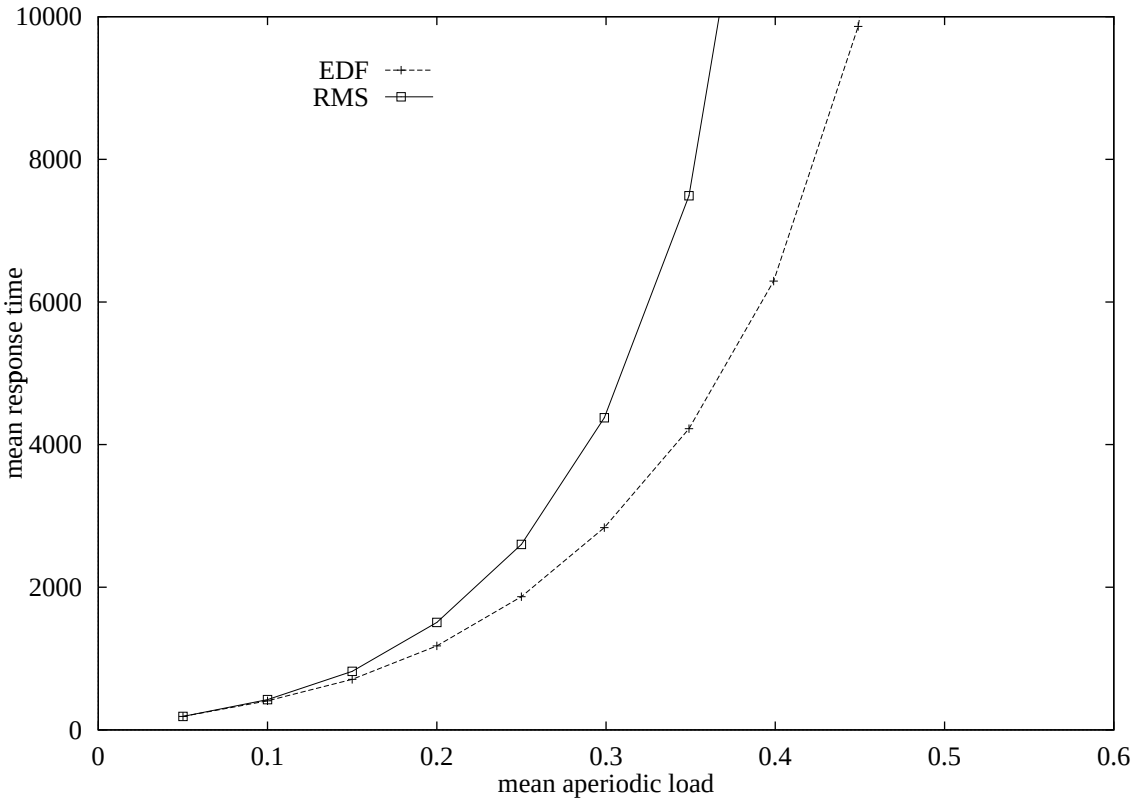

Figure 18: RM versus deadline sporadic servers, interarrival time 1800

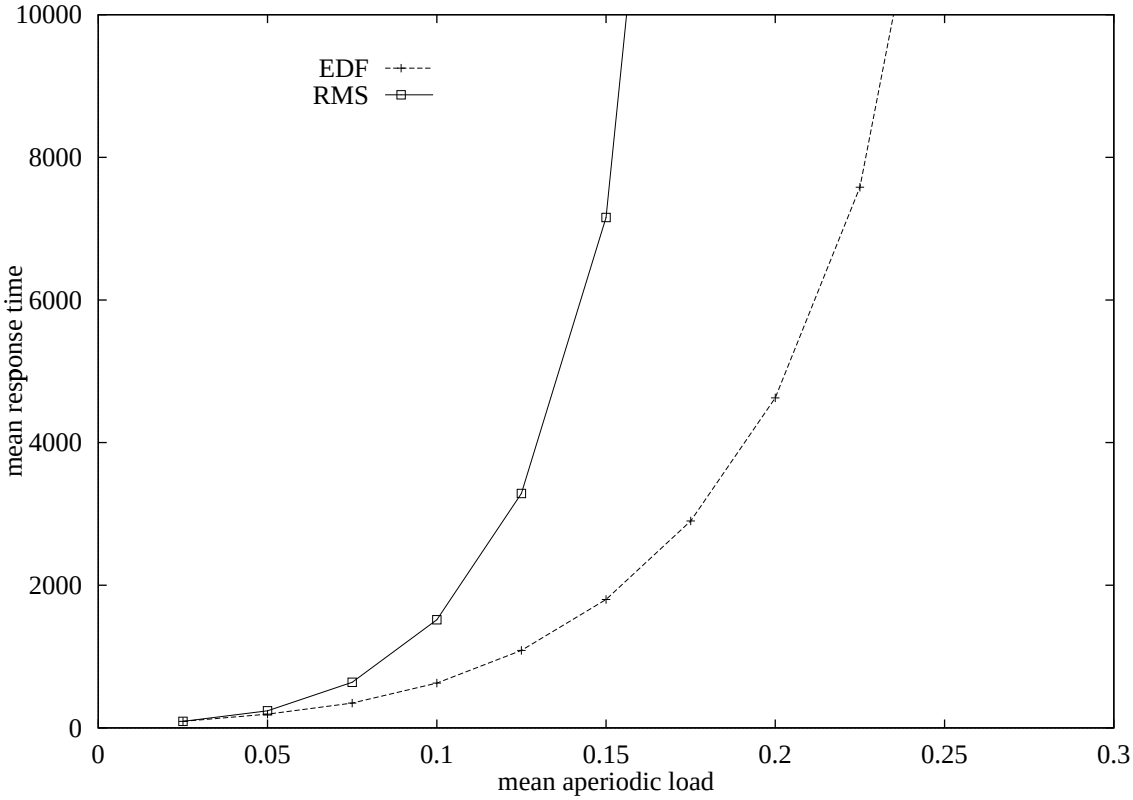

Figure 19: RM versus deadline sporadic servers, interarrival time 3600

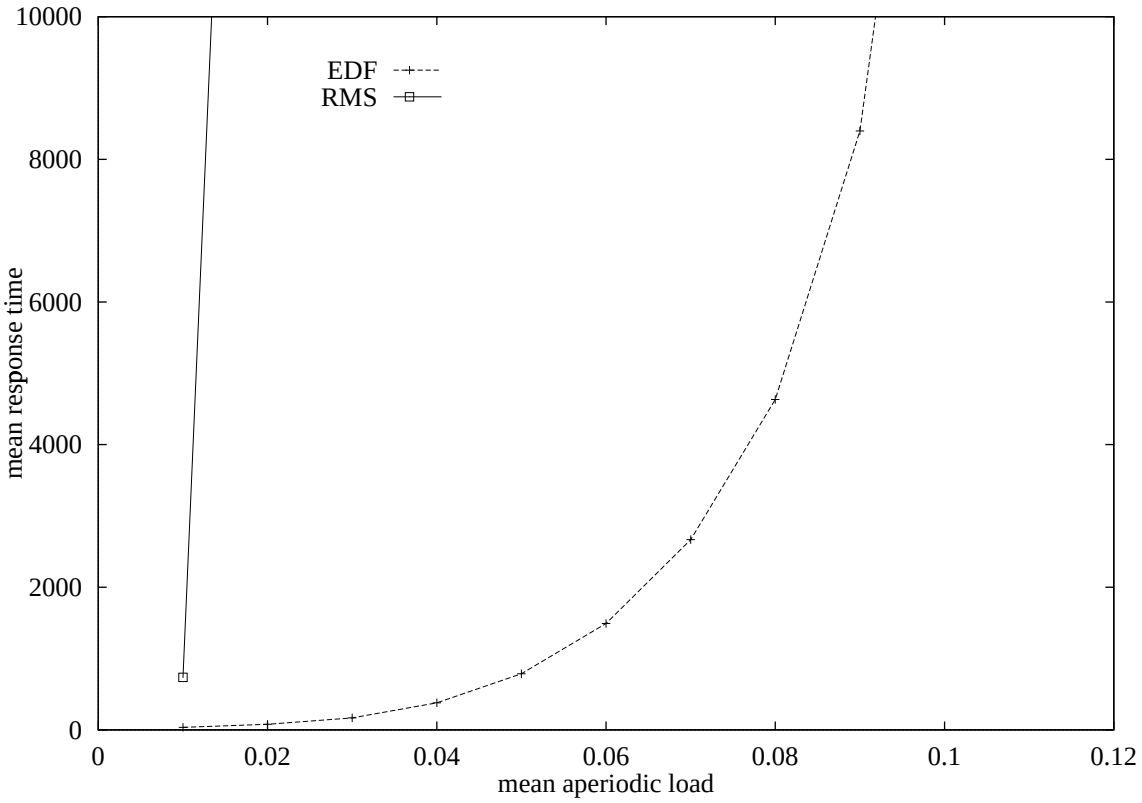

Figure 20: RM versus deadline sporadic servers, interarrival time 5400