Published in the Proceedings of the Eleventh Euromicro Conference on
Real-Time Systems, 9-11 June 1999, held at University of York, England

Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overload

Mark K. Gardner   Jane W.S. Liu
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue, MC258
Urbana, IL 61801, USA
{mkgardne, janeliu}@cs.uiuc.edu

Abstract

This paper compares the performance of three classes of scheduling algorithms for real-time systems in which jobs may overrun their allocated processor time potentially causing the system to be overloaded. The first class, which contains classical priority scheduling algorithms as exemplified by DM and EDF, provides a baseline. The second class is the Overrun Server Method which interrupts the execution of a job when it has used its allocated processor time and schedules the remaining portion as a request to an aperiodic server. The final class is the Isolation Server Method which executes each job as a request to an aperiodic server to which it has been assigned. The performance of the Overrun Server and Isolation Server Methods are worse, in general, than the performance of the baseline algorithms on independent workloads. However, under the dependent workloads considered, the performance of the Isolation Server Method, using a server per task scheduled according to EDF, was significantly better than the performance of classical EDF.

1. Introduction

In a *hard real-time system*, all jobs must meet their deadlines with a missed deadline being treated as a fatal fault. Hence hard real-time systems, which are most often found in safety or mission critical applications, are designed to ensure that there are no missed deadlines often at the expense of resource utilization and average performance. A much larger class of applications, known as *soft real-time systems*, allows some jobs to miss their deadlines in order to improve resource usage or average performance. However, unlike non-real-time systems, the goal of a soft real-time system is to meet as many deadlines as possible before attempting to maximize the average performance. As real-time systems become more pervasive, it is clear that techniques for scheduling hard, soft and non-real-time workloads on the same system are needed.

In this paper, we address the problem of scheduling jobs that may overrun their processor allocation, potentially causing the system to be overloaded. Ideally, those jobs which do not overrun are guaranteed to meet their deadlines while those which do are scheduled so as to have good response times. The scheduling algorithms we describe enable the coexistence of hard, soft and non-real-time workloads on a single system.

Our algorithms are based on an extension to the *periodic task model* [6]. According to the model, a real-time system consists of a set of *tasks*, each of which is a (possibly) infinite stream of computations or communications, called *jobs*. The execution time of a job is the amount of time the job takes to complete if it executes alone. All the jobs in a task have a common maximum execution time and are released for execution (i.e., arrive) with a common minimum inter-release time. The minimum inter-release time is called the *period* of the task and is larger than zero. A job becomes ready for execution at its release time and must complete execution by its absolute deadline or it is said to have missed its deadline. The length of time between the release time and absolute deadline of every job in a task is constant and is called the *relative deadline* of the task. The maximum utilization of a task is the ratio of the maximum execution time to the minimum period and the maximum utilization of the system is the sum of the maximum utilizations of its tasks. Finally, the release time of the first job in a task is called the *phase* of the task. We say that tasks are *in-phase* when they have identical phases.

In modern real-time systems, tasks are scheduled in a priority driven manner. At any point in time, the ready job with the highest priority executes. Most systems use a fixed priority assignment according to which all jobs in a task have the same priority. Examples of fixed priority policies are *Rate Monotonic* (RM) [6] or *Deadline Monotonic* (DM) [5]. The priority of a task under RM is proportional to the rate at which jobs in the task are released, while the priority of a task under DM is inversely proportional to the relative deadline of the task. Priorities may also be assigned dynamically. The most common dynamic priority scheduling policy is *Earliest Deadline First* (EDF) [6] which assigns priorities to jobs in order of their absolute deadlines.

Hard real-time scheduling theory determines the schedulability of a system based on the maximum execution times of the tasks in the system. In order to ensure that a system is *schedulable* (i.e., every task will meet all its deadlines), the processor bandwidth set aside for each task is equal to its maximum utilization. Because the execution times of jobs in many soft real-time systems vary widely, designing a soft real-time system using hard real-time scheduling theory often yields a system whose average utilization is unacceptably low. Instead of a maximum execution time for each task, we require that *guaranteed execution times* be specified. The guaranteed execution time is zero for non-real-time tasks, equal to the maximum execution time of any job of the task for each hard real-time task, and somewhere in between for soft real-time tasks. The schedulability of the system is determined based upon the guaranteed execution time of every task. All jobs with execution times less than or equal to the guaranteed execution time of the task will meet their deadlines. The system tries to minimize the response time of each job whose execution time exceeds its guaranteed execution times. With this modification to the periodic task model, systems containing hard, soft and non-real-time tasks can be described in a uniform manner.

A job is said to *overrun* when it executes for more than its guaranteed execution time. Depending on the amount of time available, a system may be able to schedule the remaining portion of an overrunning job so that it completes by its deadline. We say that a system is *overloaded* when it is not schedulable on the basis of the maximum execution times of its tasks and hence it is likely that some jobs will miss their deadlines. Jobs in a system may overrun without the system being overloaded; however, an overloaded system implies that at least one job overruns.

We begin in Section 2 by describing three classes of algorithms for scheduling systems with jobs that overrun. In Section 3 we describe the performance criteria used to compare the algorithms and present the results of the comparison in Sections 4, 5, and 6. In Section 7 we discuss the relationship of the algorithms to other related work. In Section 8 we summarize the findings of this study.

2. Algorithms for Scheduling Overruns

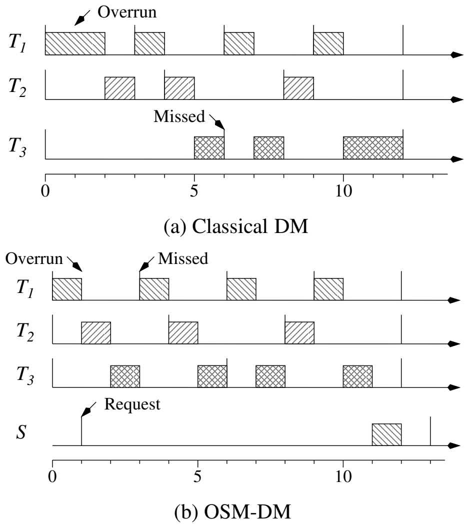
Any algorithm for scheduling jobs with a potential for overrun must meet two criteria if it is to perform well. First, it must guarantee that jobs which do not overrun meet their deadlines. Second, it should maximize the number of overrunning jobs that meet their deadlines or minimize the response times of overrunning jobs. In this section, we discuss two classes of algorithms, the first of which achieves the former while striving for the latter. The second relaxes the guarantee that non-overrunning jobs will meet their deadlines in order to perform better on dependent workloads.

As a basis of comparison, we will also consider the performance of classical fixed and dynamic priority hard real-time scheduling algorithms under various degrees of overload. The algorithms used as a baseline are the Deadline Monotonic (DM) [5] and Earliest Deadline First (EDF) [6] algorithms. These algorithms have been shown to be optimal in that DM and EDF will ensure that no deadlines are missed if all the tasks can be scheduled without missing a deadline by any fixed or dynamic priority algorithm, respectively. The DM algorithm also has the desirable property that an overrun will not cause a job of a higher priority task to miss its deadline. In contrast, it is well known that EDF behaves unpredictably upon overrun. However, EDF has a higher schedulable utilization than DM and hence makes better use of resources as long as the maximum utilization is no greater than one. The desire to combine the predictability of DM with the optimal schedulable utilization of EDF motivates the two classes of algorithms that follow.

The second class of algorithms, called the Overrun Server Method (OSM), is both a simplification and an extension of the Task Transform Method proposed by Tia *et al.* [12]. Under OSM, a job is released for execution and scheduled in the same manner as it would be according to algorithms in the baseline class. At the time of overrun, the execution of the job is interrupted and the remaining part of the job is released as an aperiodic request to a server. A Sporadic Server (SS) [8] is used to execute the requests in fixed priority systems while either a Constant Utilization Server (CUS) [1] or Total Bandwidth Server (TBS) [9] is used to execute requests in a dynamic priority system. We denote the former as OSM-DM and the latter as OSM-EDF.

Sporadic Servers are specified by a replenishment period and an execution budget. They have the property that they demand no more time than a corresponding periodic task with the same period and maximum execution time. Thus, the schedulability of a system containing Sporadic Servers can be determined by the methods applicable to systems of periodic tasks. We use a particularly aggressive Sporadic Server implementation which reduces the average response time of requests [11]. A Sporadic Server has a separate ready queue so it makes sense to consider various queueing disciplines. We consider the First-Come-First-Served (FCFS), Deadline Monotonic (DM), and *Shortest time Remaining at Overrun* (SRO) queue disciplines. (In the latter, the priority of a request is inversely proportional to the amount of work remaining at the time of overrun.)

As an example of OSM-DM, consider a system of three tasks. Task T_1 has a period of 3 and a guaranteed execution time of 1. Task T_2 has a period of 4 and a guaranteed execution time of 1. Task T_3 has a period of 6 and a guaranteed execution time of 2. In Figure 1(a), one job of T_1 overruns. As a result, T_3 misses a deadline at 6 when the tasks are scheduled according to the DM algorithm. Figure1(b) shows the same workload but with a Sporadic Server having a replenishment period of 12 and an execution budget of 1. According to classical schedulability theory, the three



(a) Classical DM

(b) OSM-DM

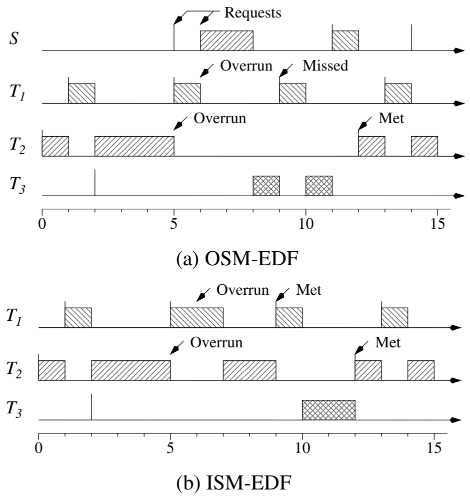
Figure 1. Behavior of DM upon overrun

tasks and the server are schedulable according to DM on the basis of guaranteed execution times. Note that now the jobs of T_3 meet their deadlines in spite of the overrun by the first job in T_1.

The schedulability of an OSM-EDF system using either CUS or TBS can also be determined by the methods applicable to systems of period tasks because they demand no more time than a corresponding task with the same utilization. The difference between the two algorithms is that TBS uses background time whereas CUS does not. Typically, CUS is preferred when there are several servers and it is undesirable for the servers to compete for background time. However, one would expect the average response time of requests executed by CUS to be greater than if executed by TBS. We later show results that support this conclusion.

The final class of algorithms is the Isolation Server Method (ISM) named for its use of SS, CUS, or TBS to isolate other parts of the system from the behavior of an overrunning job. Jobs are submitted as aperiodic requests to the server assigned to their task at the time of their release and execute completely under server control. A server may be assigned to execute jobs from multiple tasks. Whereas OSM requires a portion of the processor separate from the tasks to be allocated to servers, ISM allocates portions of the processor to servers and does not allocate time to the tasks. Because jobs of a task are released as requests to a server, an overrunning job under ISM can only delay the completion of jobs from tasks assigned to that server. The execution of jobs in tasks not assigned to that server are isolated from the overrunning job. Unlike OSM, ISM cannot guarantee all jobs with execution times which do not exceed their guaranteed execution times will complete by their deadlines. We denote fixed priority ISM by ISM-DM and dynamic priority ISM by ISM-EDF.

As an example, consider another system of three tasks scheduled according to the EDF algorithm together with



(a) OSM-EDF

(b) ISM-EDF

Figure 2. Behavior of server upon overrun

TBS. Task T_1 has a period of 4 and a guaranteed execution time of 1. Task T_2 has a period of 12 and a guaranteed execution time of 4. Task T_3 has a period of 24 and a guaranteed execution time of 2. Under OSM-EDF, the overrun server has a replenishment period of 3 and an execution budget of 1 while ISM-EDF has a separate server per task with utilizations equal to the utilizations of the respective tasks. According to classical schedulability theory, the system is schedulable under both OSM-EDF and ISM-EDF. Suppose that a job of T_2 overruns by 2 at time 6 and a job of T_1 overruns by 1 at time 7, as shown in Figure 2. The overrunning job of T_1 misses its deadlines under OSM-EDF but completes in time under ISM-EDF. Also, both overrunning jobs complete earlier under ISM-EDF than under OSM-EDF because ISM servers execute jobs at their original priority, while OSM servers execute the overrun portion of jobs at the priority of the corresponding server. In this case, the server has a lower priority than T_1 while it executes the overrunning job of T_2.

3. Comparison Study

In this section, we describe the methodology used to compare the performance of the three classes of algorithms. The average performance of the system is obtained by discrete event simulation. We first discuss the criteria used to compare the performance and then describe the workload used in the comparison.

3.1. Performance Criteria

In Section 2 we stated what the ideal behavior of an algorithm for scheduling jobs in the presence overruns should be. First and foremost is the requirement that jobs which do not exceed their guaranteed execution times should never miss a deadline. OSM meets this condition by design. ISM relaxes this condition slightly in that an overrunning job may delay the completion of subsequent jobs assigned to the same server. The next condition is that the algorithm maximizes the number of deadlines met and minimizes the response time of overrunning jobs. Thus, the fraction of deadlines met by jobs in each task and the average response time of jobs in a task are important metrics.

It is clear that the above metrics cannot be directly compared for different workloads because, for example, the percentage of deadlines missed depends upon the execution time and deadline of the tasks, both of which vary across workloads. for this reason, we use the ratio of a metric measured for one algorithm against the same metric measured for another algorithm on the same workload, averaged over all the workloads. For example, if the average ratio of deadlines met is 1.0, then the two algorithms have equivalent performance on the average. Likewise, an average ratio of deadlines met of 1.25 indicates that the first algorithm performs 25% better than the second on the average. Since we wish to compare the performance of the algorithms on a system-wide basis rather than a task by task basis, the average of the ratios of all the tasks were taken.

3.2. Workload Generation

The performance of a scheduling algorithm depends upon the average utilization of the system. At low utilizations, sufficient time exists for nearly all overrunning jobs to complete in time. As the utilization increases, overrunning jobs become more likely to miss their deadlines. At some point, overruns will cause the system to be overloaded. As long as the average utilization of the system is less than one, the system will continue to function, albeit with increasingly reduced performance. For the workloads used in this study, an average utilization of 0.50 is sufficient for nearly all jobs to met their deadlines. The average utilizations we consider are 0.50, 0.75, 0.90 and 0.95.

Given the average utilization of the system, the average utilization of a task is obtained by multiplying the average system utilization by a utilization factor for the task. The utilization factors of the tasks in a system are uniformly distributed in the range $(0, 1]$ and normalized such that the sum of the factors equals 1.0. The execution time of jobs in a task are random variables with a common distribution, either uniform or exponential. The mean execution time of a job is equal to the mean utilization of its task multiplied by the (constant) period of the task. The minimum execution time is 1 time unit. The periods of the tasks are a constant and are uniformly distributed in the range $[1000, 10000]$.

Overruns are often caused by common factors and hence the execution time of jobs are likely to be correlated. For dependent execution times, we model dependencies as being exclusively between jobs in a task and following a fixed pattern. The pattern is represented by a list of execution time factors, with a mean value of 1.0, representing the correlation between the execution times of consecutive jobs. For example, suppose that the pattern is $\{2, 0.5, 1, 0.5\}$. If the mean execution time of a set of dependent jobs is 100, the actual execution times of the jobs are $\{200, 50, 100, 50\}$. (The mean execution time of a set of dependent jobs is computed by the previous procedure.) Dependence patterns are varied and clearly application dependent. In our current study, we examined the performance of the algorithms for the (independent) pattern $\{1\}$ and for the (dependent) pattern $\{1, 1\}$.

For each average system utilization and distribution type, we generate 100 systems, each consisting of 8 tasks. Each of the tasks in a system has a minimum of 2000 jobs.

4. Baseline

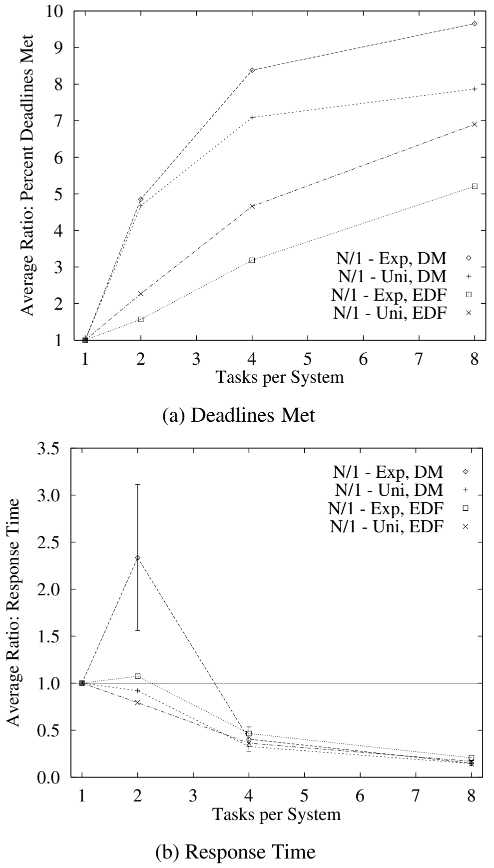
We compare the performance metrics for equal release times (in-phase release) and random release times. For the random release time case, 30 runs with the initial release time of each task uniformly distributed in $[0, P_i]$ were performed. The results clearly indicate that the average performance of the system does not depend on the initial phase of the tasks. (We note that while the average ratios are not sensitive to the initial phase of the tasks, the minimum and maximum ratios are.)

Another factor which may influence the performance is the number of tasks in the system. Figure 3 gives ratios of the performance of systems with various numbers tasks to the performance of systems with one task. The average system utilization is 95%. In general, the average performance of the system improves with increased numbers of tasks. However, Figure 3(b) shows the average response time ratio of systems with two tasks is worse than systems with one task for an exponential workload scheduled DM. There are two factors which cause this behavior. First, distributing a given system load across more tasks lowers the average overrun per task thereby improving the average response time ratio. Second, more than one task in a system implies that a job may be delayed by the execution of another job causing its response time to increase. This negative effect is most pronounced with two tasks per system because the average execution time per task is greatest. The latter effect is most prevalent for an exponential distribution since its average overrun is greater than it is for a uniform distribution with the same mean and minimum. We use 8 tasks per system for the remainder of the paper.

Finally, we compare the performance of EDF and DM on independent and dependent workloads. As can be seen in Figure 4, the average performance of the EDF policy is clearly worse than the average performance of DM.

5. Overrun Server Method

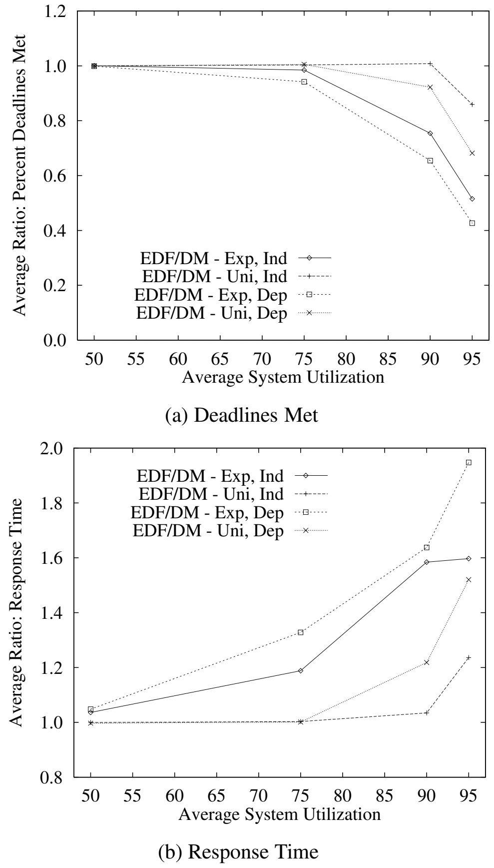
As discussed above, one way to schedule jobs with the potential for overrun is to suspended a job when it has executed for its guaranteed execution time and release the remainder of the job as a request to an overrun server. Only jobs with execution times in excess of their guaranteed values may miss their deadlines. The main issues in using overrun servers to schedule jobs are the number of servers



(a) Deadlines Met

(b) Response Time

Figure 3. Performance vs. number of tasks



(a) Deadlines Met

(b) Response Time

Figure 4. EDF vs. DM

to use, the assignment of jobs to servers, the parameters of each server, and the queueing discipline employed by the server.

5.1. Number of Servers

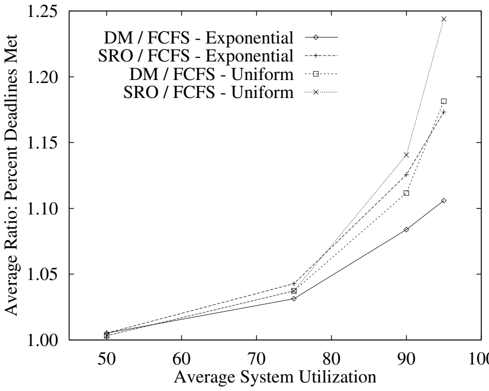
The number of servers can range from a single server for all tasks to a server for each overrunning task. Clearly the aggregate average utilization of the servers can be at most $1 - \sum_i U_i$, where U_i is the average utilization of task T_i, without the potential for causing non-overrunning jobs to miss their deadlines. (It is less than this under fixed priority scheduling policies due to a lower schedulable utilization.) The question is how to distribute the uncommitted utilization of processor. If the remaining utilization is divided without regard for the average utilization of overruns assigned to each server, e.g., dividing the remaining utilization evenly, some servers may have a proportionally greater average demand than others. This utilization distribution is likely to cause jobs to have worse response times and be less likely to complete by their deadlines. On the other hand, dividing the remaining utilization proportionally according to the average utilization of overruns assigned to each server ensures that servers will receive processor time proportional to their load.

The results indicate that the performance of a single server for all tasks and a server per task bound the performance of systems with intermediate numbers of servers. In addition, there is no significant difference in performance between different assignments of tasks to servers since the load on the servers are equal. For brevity, we present only the results for systems with 1 or 8 servers having groups of consecutive tasks assigned to servers.

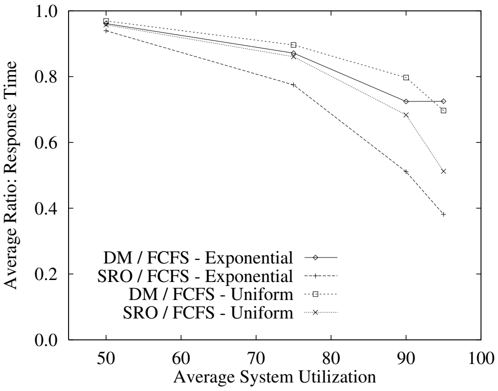
5.2. Deadline Monotonic

Given the server utilization, an overrun server in a fixed priority system requires the specification of a replenishment period in order to establish a budget for the server. It also requires a scheduling discipline for its job queue. For simplicity, we consider the choice of replenishment period and queue discipline for a system with a single overrun server.

The literature pertaining to the service of aperiodic requests almost universally suggests that the priority of a server should be greater than that of the tasks. To determine the best server period, we simulate the behavior of a simplified set of 30 systems consisting of 3 tasks with 1000 jobs per task and 9 priorities chosen to be slightly higher than, equal to, and slightly lower than the priority of each task. The execution time of jobs in the highest priority are

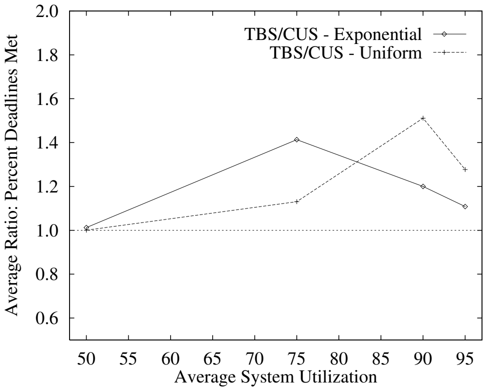


(a) Deadlines Met

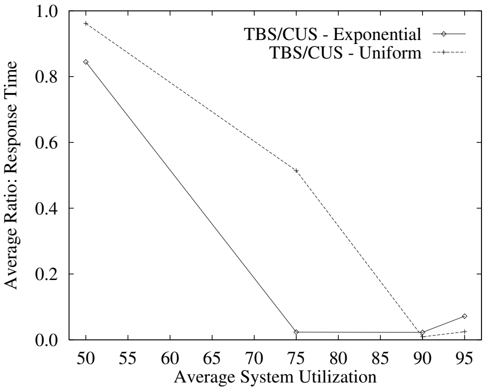


(b) Response Time

Figure 5. Performance of queue disciplines



(a) Deadlines Met



(b) Response Time

Figure 6. Performance of TBS vs. CUS

taken from a uniform or exponential distribution. The execution times of the other tasks are constant. The task periods are in the range $[100, 1000]$. The average utilizations of the systems are 75%, chosen to be less than the Liu and Layland bound of 75.7% guaranteeing that the systems are be schedulable on the basis of mean execution times. The results indicate that the choice of Sporadic Server priority does not affect the average response time or percentage of deadlines met.

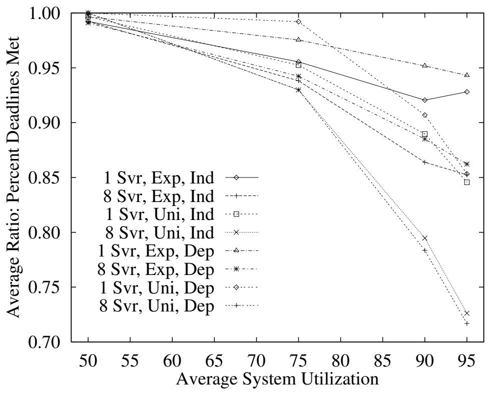
Next we consider the choice of queue discipline for the Sporadic Server ready queue. We use FCFS as a baseline (as in [12]) and compare its performance with DM and SRO on the workloads discussed in Section 3.2. Figure 5 shows that the percentage of deadlines met when DM is used as the queue discipline is higher than when FCFS is used. Likewise, more deadlines are met when SRO is used.

The latter result may be surprising because SRO performs poorly as a real-time scheduling algorithm since it prioritizes jobs on the basis of remaining work at release rather than on task deadlines or periods. However, it minimizes response time and thus it allows more overrunning jobs to complete by their deadlines when used as a SS queue discipline. We use SRO as the queue discipline in the experiments that follow.

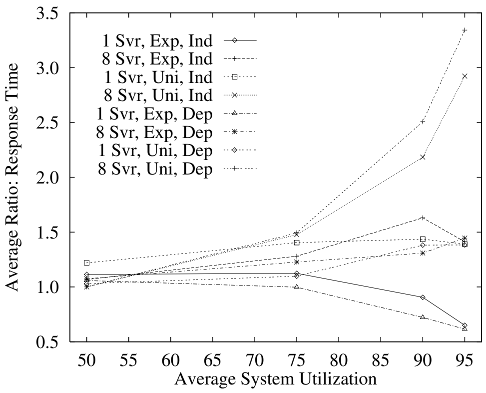
5.3. Earliest Deadline First

We now consider OSM in a deadline-driven system. As stated earlier, we consider either CUS or TBS in connection with an EDF scheduling policy. The CUS and TBS overrun servers are specified by establishing server utilizations and assigning tasks to servers in the manner discussed earlier.

Again, a CUS server and a TBS server behave identically except that the latter is allowed to compete for background processing time. Because of this, jobs executed by a TBS server may complete earlier and meet their deadlines. Thus we would expect TBS to perform better than CUS as an overrun server. As Figure 6 shows, the average response time of TBS is much lower than CUS, particularly at moderate to high average system utilization. The percentage of deadlines met is also greater for TBS. The concave shape of the curves comes from two opposing effects. At low load, the average overrun is small so TBS and CUS perform similarly. Increasing load causes an increase in the average overrun which TBS is able to execute earlier. At the same time, increasing the load decreases the amount of background time that TBS can exploit. As the average load approaches 100%, the performance once again becomes similar because the algorithms behave similarly. We use TBS as the server for the remainder of the paper.

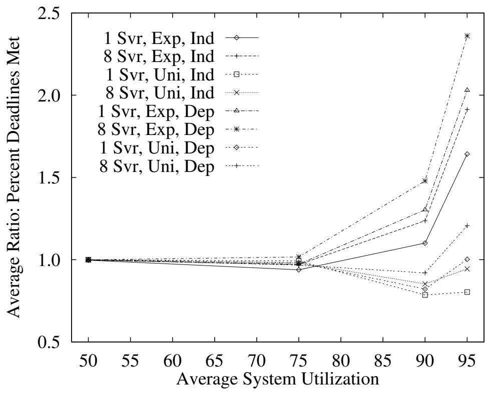


(a) Deadlines Met

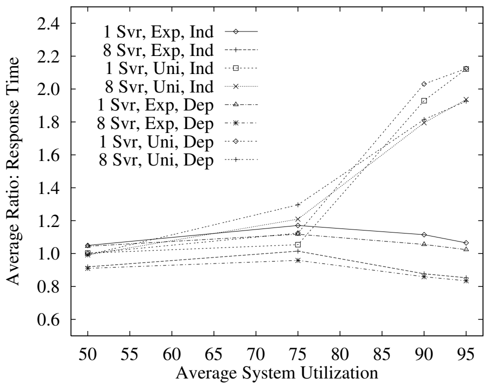


(b) Response Time

Figure 7. OSM-DM vs. DM



(a) Deadlines Met



(b) Response Time

Figure 8. OSM-EDF vs. EDF

5.4. Performance of OSM vs. Baseline

Figures 7 and 8 compare the performance of OSM with the performance of the baseline algorithms. As Figure 7(a) shows, significantly more deadlines were missed by OSM-DM. The Liu and Layland bound for the baseline algorithm is 72.4% and hence the systems are not likely to be schedulable at 75% average system utilization or above. Adding overrun servers reduces the likelihood of being schedulable even further. In spite this, we present results for higher utilizations as an indication of soft real-time behavior at high system loads. Figure 7(b) indicates that OSM-DM yields better response times than classical DM for a single overrun server and dependent exponential workloads. In all cases, a single overrun server for all tasks performs better than having an overrun server per task because SS cannot reclaim the time allocated to another server even though the server is idle.

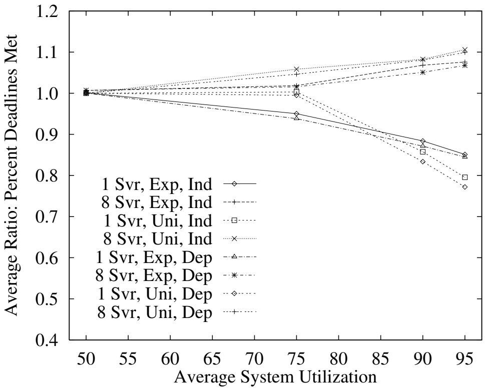
In Figure 8 we see that OSM-EDF performs better than classical EDF for exponential workloads, particularly if the workload has dependencies. Also it performs better on exponential workloads than on uniform ones because the probability of overrun is less for an exponential distribution than for a uniform one when the guaranteed execution time equals the mean. The average amount of processor time not used by jobs which do not overrun is greater than the average amount of overrun for an exponential distribution, whereas they are equal for a uniform distribution with the same mean and minimum. Finally, we observe that OSM-EDF performs better with a server per task than with a single server because TBS makes use of background time unused by other servers.

5.5. Performance of OSM-EDF vs. OSM-DM

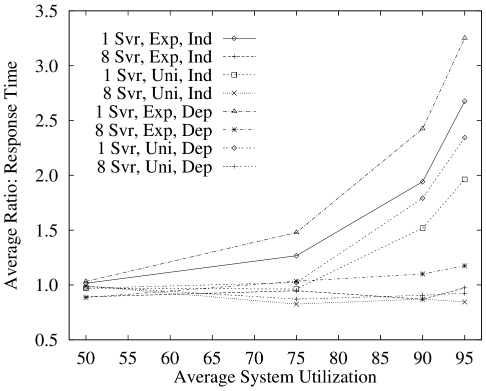
Under classical scheduling, DM performs better than EDF when jobs overrun. However, as Figure 9 shows, OSM-EDF generally performs better than OSM-DM in the server per task configuration. Thus, in general, OSM should employ a server per task and be scheduled EDF.

6. Isolation Server Method

Another way to schedule jobs with the potential for overrun is to release the jobs as requests to isolation servers for execution. The same issues need to be considered when scheduling jobs using ISM as when scheduling overruns using OSM: the number of servers, the assignment of jobs to servers, the budget of each server, and the queueing discipline employed by each server.

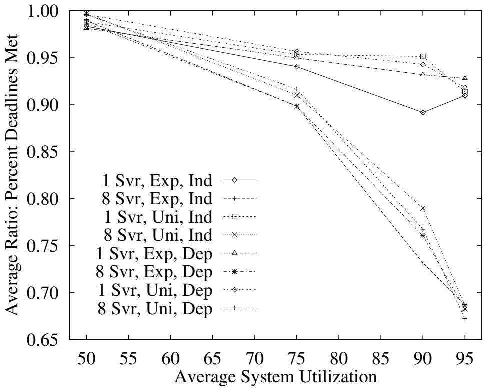


(a) Deadlines Met

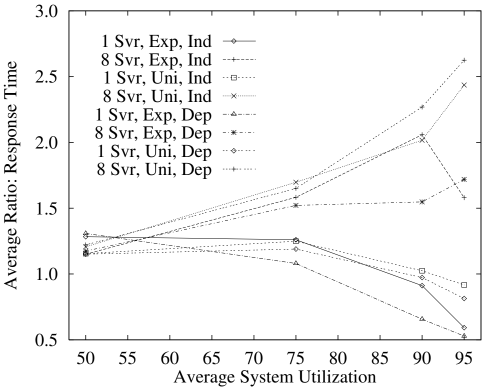


(b) Response Time

Figure 9. OSM-EDF vs. OSM-DM



(a) Deadlines Met



(b) Response Time

Figure 10. ISM-DM vs. DM

6.1. Effects of Parameters

As in Section 5, we consider 1–8 servers. Unlike OSM, each server is allocated a portion of the processor bandwidth proportional to the fraction of the total utilization the tasks it executes contribute. As is the case for OSM-DM, the performance metrics are not sensitive to the choice of SS server replenishment period using ISM-DM, and the SRO queue discipline gives the best results. TBS is also clearly superior to CUS for ISM-EDF. Here also, the performance of a single server for all tasks and a server per task bounds the performance of intermediate numbers of servers hence only the results for 1 and 8 servers are presented.

6.2. Performance of ISM vs. Baseline

Figures 10 and 11 compare the performance of ISM with the performance of the baseline algorithms. As Figure 10 shows, the performance of ISM-DM is worse than classical DM; the systems are less likely to be schedulable as the average system utilization increases. We note that ISM-DM (in Figure 10(b)) has a higher average response ratio than OSM-DM (in Figure 7) at 50% utilization because overrunning jobs may delay subsequent jobs under ISM where they cannot under OSM. ISM-DM has a slightly better response ratio than OSM-DM at high utilizations, however. Once again, ISM-DM gives the best results with one server for all tasks for the same reason that OSM-DM does.

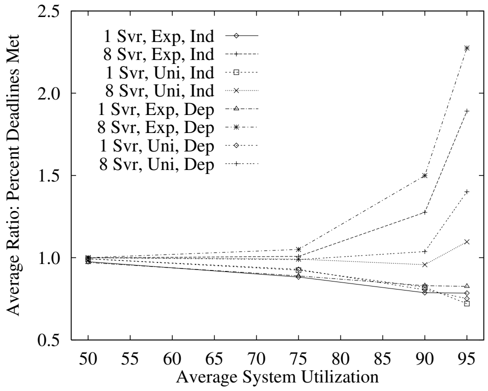
In Figure 11(a) we see that ISM-EDF with one server per task meets more of its deadlines on the average than classical EDF and less with a single server for all tasks. Also, the average response time ratio is better for a server per task, as Figure 11(b) shows. This behavior is evident for both dependent and independent workloads with exponential or uniform distributions.

6.3. Performance of ISM-EDF vs. ISM-DM

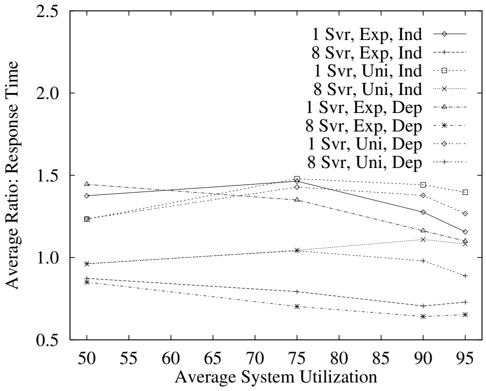
As Figure 12 shows, the performance of ISM-EDF is better than the performance of ISM-DM in the server per task configuration and worse in the single server configuration. There was surprisingly little variation between independent and dependent workloads. For the best performance, ISM should employ a server per task and be scheduled according to the EDF algorithm.

7. Related Work

As was mentioned earlier, the Overrun Server Algorithm (OSM) is a simplification and extension of the Task Transform Method proposed by Tia *et al.* [12]. Like their work, OSM also transforms a job into a mandatory periodic task,

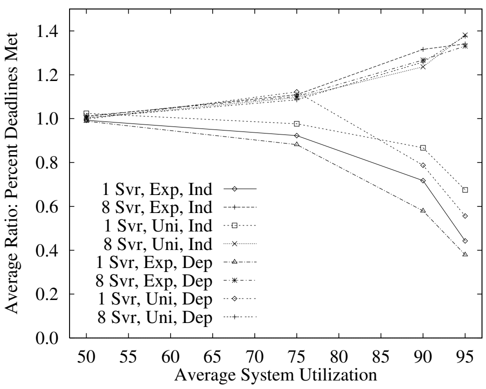


(a) Deadlines Met

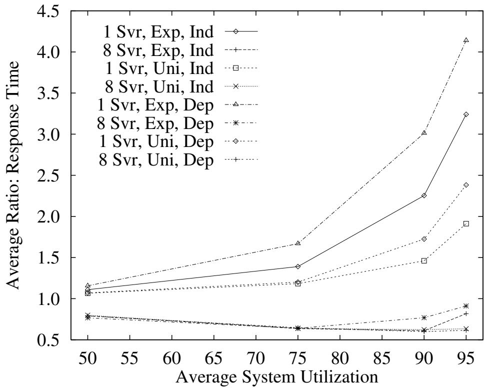


(b) Response Time

Figure 11. ISM-EDF vs. EDF



(a) Deadlines Met



(b) Response Time

Figure 12. ISM-EDF vs. ISM-DM

whose maximum execution time is the guaranteed execution time in our model, and a request to a server for the execution of the remaining portion. Under a fixed priority scheduler, the remaining portion is executed by a Sporadic Server [8]. Under an EDF scheduler, requests are executed by either a Constant Utilization Server [1] or a Total Bandwidth Server [9] rather than a Slack Stealer [4].

The Isolation Server Method (ISM) is similar to the Proportional Share Resource Allocation algorithm (PSRA) [10]. Both assign a portion of the processor bandwidth to a task. Whereas PSRA allocates the assigned portion to jobs in discrete-sized time quanta, ISM allocates the portion in variable sized chunks defined through preemption by higher priority jobs. The difference between the portion of a processor bandwidth a task receives under PSRA and the ideal is bounded by a constant equal to the quantum chosen. ISM provides the ideal portion precisely. Both algorithms allow the integration of real and non-real-time processing, are easy to implement and prevent ill-effects of overrunning jobs on jobs in other tasks.

The problem of assigning n fixed priority tasks to m servers, where $m \leq n$, is addressed in [3]. They give an exponential time algorithm for determining the assignment that gives the smallest response time while ensuring that the system remains schedulable, if such an assignment exists. On the other hand, the systems studied in this work are, for the most part, not schedulable according to hard real-time schedulability theory. In addition, we relax the assumption that jobs are served FCFS. Instead, we also consider the behavior of Sporadic Servers with a fixed priority queue discipline (DM) and a queue discipline designed to minimize response times (SRO). Our results indicate that the behavior of a system with multiple servers is bounded by the assignment of all tasks to a single server and the assignment of each task to its own server.

In [2], Ghazalie and Baker present the performance of several aperiodic servers in a deadline-driven environment. Their focus is on the scheduling of aperiodic tasks while our focus is on scheduling overruns using servers. One of the servers they consider is a variation of the Sporadic Server, adapted to a dynamic priority environment, while we use Sporadic Servers under fixed priority and employ either a Constant Utilization Server or a Total Bandwidth Server under dynamic priority. They observed that the average response time of an aperiodic task decreases with increases in Sporadic Server server period (dynamic priority) while we observed that the average performance did not change with the server period (fixed priority). Besides the differences due to fixed versus dynamic priority, the difference is likely the result of considering the behavior of a single system versus the average behavior of many systems. We also considered three disciplines for prioritizing the server ready queue while it appears that their server executes requests in order of arrival. Finally, we consider dependencies between the execution times of consecutive jobs.

Another closely related work is the Open Systems Environment (OSE) described in [1]. Similar to ISM-EDF, OSE ensures that the behavior of a task does not interfere with the ability of other tasks to meet their deadlines. The primary motivation of OSE is to allow real-time applications to be developed and validated independently by assuming that each application runs alone on a slow processor and then are executed together on a fast processor without causing missed deadlines. The primary motivation of OSM and ISM is to accommodate overrun. Thus, OSM and ISM are complimentary to OSE.

The work described in this paper differs from recent approaches, such as [7], by relaxing hard real-time constraints in a controlled manner rather than attempting to add support for real-time tasks without compromising fairness. The latter approach is unable to provide any form of hard real-time guarantee under overload by virtue of an insistence on being fair on the average. In contrast, the algorithms discussed in this paper are able to make real-time guarantees and thereby sacrifice the ability to be fair to all tasks on the average. However, fairness within non-real-time tasks can achieved without sacrificing the ability to make guarantees by executing non-real-time tasks within a server which implements a traditional time-share scheduler. Thus the algorithms described in this paper are able to guarantee the deadlines of hard real-time tasks (and soft real-time tasks whose jobs do not exceed the guaranteed execution time) while scheduling non-real-time tasks fairly within the portion of the processing time allocated for them.

8. Conclusions

Modern real-time systems are increasingly being required to accommodate mixed hard, soft and non-real-time workloads. Designing such systems according to classical schedulability theory can yield systems with low resource utilization and poor average performance. We have extended the periodic task model by requiring that guaranteed execution times for tasks be supplied rather than the maximum execution time of all jobs in each task. This modification allows us to uniformly describe hard, soft and non-real-time tasks within the same framework. We say that a job overruns if it exceeds its guaranteed execution time and have defined two classes of algorithms for scheduling such jobs. We have also explored the performance of the two methods for various workloads in comparison to the class of classical hard real-time scheduling algorithms.

In general, the classical algorithms perform better when jobs overrun than do OSM and ISM on independent workloads. However, the causes of overrun are often correlated causing dependencies between jobs. For the dependent workload considered, ISM with a server per task scheduled according to EDF performed better than classical EDF. Additional work needs to be done to further understand the behavior of systems on workloads with more complex dependencies between jobs in the same task and on workloads with dependencies between jobs in different tasks.

Although we have explicitly considered the behavior of static systems in this paper, the results are also applicable to systems in which tasks arrive and leave ensuring that non-overrunning jobs will meet their deadlines through the use of admission control.

References

[1] Z. Deng, J. W.-S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of the Ninth Euromicro Workshop on Real-Time System*, pages 191–199, Toledo, Spain, June 1997.

[2] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1):31–67, July 1995.

[3] D. I. Katcher, S. S. Sathaye, and J. K. Strosnider. Fixed priority scheduling with limited priority levels. *IEEE Transactions on Computers*, 44(9):1140–1144, Sept. 1995.

[4] J. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *IEEE Real-Time Systems Symposium*, pages 110–123, Dec. 1992.

[5] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[7] J. Nieh and M. S. Lam. Smart: A processor scheduler for multimedia applications. In *Proceedings of SOSP-15*, page 223, Dec. 1995.

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

[9] M. Spuri, G. Buttazzo, and F. Sensini. Robust aperiodic scheduling under dynamic priority systems. In *In Proceedings of the 17th Real-Time System Symposium*, pages 210–219, Dec. 1996.

[10] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *In Proceedings of the 17th Real-Time System Symposium*, Dec. 1996.

[11] J. Sun. *Fixed Priority Scheduling to Meet End-to-End Deadlines in Distributed Real-Time Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[12] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 164–173, Chicago, Illinois, May 1995. IEEE.