A Non-preemptive Scheduling Algorithm for Soft Real-time Systems

Wenming Li, Krishna Kavi[1] and Robert Akl
The University of North Texas

Abstract

Real-time systems are often designed using preemptive scheduling and worst-case execution time estimates to guarantee the execution of high priority tasks. There is, however, an interest in exploring non-preemptive scheduling models for real-time systems, particularly for soft real-time multimedia applications. In this paper, we propose a new algorithm that uses multiple scheduling strategies for efficient non-preemptive scheduling of tasks. Our goal is to improve the success ratio of the well-known *Earliest Deadline First* (EDF) approach when the load on the system is very high and to improve the overall performance in both underloaded and overloaded conditions. Our approach, known as *group-EDF* (gEDF) is based on dynamic grouping of tasks with deadlines that are very close to each other, and using *Shortest Job First* (SJF) technique to schedule tasks within the group. We will present results comparing gEDF with other real-time algorithms including, EDF, Best-effort, and Guarantee, by using randomly generated tasks with varying execution times, release times, deadlines and tolerance to missing deadlines, under varying workloads. We believe that grouping tasks dynamically with similar deadlines and utilizing a secondary criteria, such as minimizing the total execution time (or other metrics such as power or resource availability that can be easily extended) for scheduling tasks within a group, can lead to new and more efficient real-time scheduling algorithms.

Keywords: Soft Real-time Systems, Non-preemptive Real-time Scheduling, Earliest Deadline First (EDF), Shortest Job First (SJF), Best-Effort Scheduling, Group EDF.

1. Introduction

The *Earliest Deadline First* (EDF) algorithm is the most widely studied scheduling algorithm for real-time systems [i]. For a set of preemptive tasks (be they periodic, aperiodic, or sporadic), EDF will find a schedule if a schedule is possible [ii]. The application of EDF for non-preemptive tasks is not as widely investigated. It is our contention that non-preemptive scheduling is more efficient, particularly for soft real-time applications and applications designed for multithreaded systems, than the preemptive approach since the non-preemptive model reduces the overhead needed for switching among tasks (or threads) [iii, iv]. EDF is optimal for sporadic non-preemptive tasks, but EDF may not find an optimal schedule for periodic and aperiodic non-preemptive tasks. It has been shown that scheduling periodic and aperiodic non-preemptive tasks is NP-hard [v, vi, vii]. However, non-preemptive EDF techniques have produced near optimal schedules for periodic and aperiodic tasks, particularly when the system is lightly loaded. When the system is overloaded, however, it has been shown that the EDF approach leads to very poor performance (low success rates) [viii]. In this paper, a system load or utilization is used to refer to the sum of the execution times of pending tasks as related to the time available to complete the tasks. The poor performance of EDF is due to the fact that, as tasks that are scheduled based on their deadlines miss their deadlines, other tasks waiting for their turn are likely to miss their deadlines also – an outcome sometimes known as the domino effect. It should also be remembered that *Worst Case Execution Time* (WCET) estimates for tasks are used in most real-time systems. We believe that, in practice, WCET estimates are very conservative, and more aggressive scheduling schemes based on average execution times for soft real-time systems using either EDF or hybrid algorithms can lead to higher performance.

While investigating scheduling algorithms, we have analyzed a variation of EDF that can improve success ratios (that is, the number of tasks that have been successfully scheduled to meet their deadlines), particularly in overloaded conditions. The new algorithm can also decrease the average response time for tasks. We call our algorithm group-EDF, or gEDF, where the tasks with "similar" deadlines are grouped together (i.e., deadlines that are very close to one another), and the *Shortest Job First* (SJF) algorithm is used for scheduling tasks within a group. It should be noted that our approach is different from adaptive schemes that switch between different scheduling strategies based on system load; gEDF is used in overloaded as well as underloaded conditions. The computational complexity of gEDF is the same as that of EDF. In this paper, we will evaluate the performance of gEDF using randomly generated tasks with varying execution times, release times, dead-

[1] Please direct all correspondence to Krishna Kavi, Department of Computer Science and Engineering, The University of North Texas, P.O. Box 311366, Denton, Texas 76203, kavi@cse.unt.edu

lines and tolerance to missing deadlines, under varying loads.

We believe that gEDF is particularly useful for soft real-time systems as well as applications known as "anytime algorithms" and "approximate algorithms," where applications generate more accurate results or rewards with increased execution times [ix, x]. Examples of such applications include search algorithms, neural-net based learning in AI, FFT and block-recursive filters used for audio and image processing. We model such applications using a tolerance parameter that describes by how much a task can miss its deadline, or by how much the task's execution time can be truncated when the deadline is approaching.

This paper is organized as follows. In section 2, we present related work. In section 3, we present our real-time system model. Numerical results are presented in section 4. Conclusions are given in section 5.

2. Related Work

The EDF algorithm schedules real-time tasks based on their deadlines. Because of its optimality for periodic, aperiodic, and sporadic preemptive tasks, its optimality for sporadic non-preemptive tasks, and its acceptable performance for periodic and aperiodic non-preemptive tasks, EDF is widely studied as a dynamic priority-driven scheduling scheme [v]. EDF is more efficient than many other scheduling algorithms, including the static *Rate-Monotonic* scheduling algorithm. For preemptive tasks, EDF is able to reach the maximum possible processor utilization when lightly loaded. Although finding an optimal schedule for periodic and aperiodic non-preemptive tasks is NP-hard [vi, vii], our experiments have shown that EDF can achieve very good results even for non-preemptive tasks when the system is lightly loaded. However, when the processor is over-loaded (i.e., the combined requirements of pending tasks exceed the capabilities of the system) EDF performs poorly. Researchers have proposed several adaptive techniques for handling heavily loaded situations, but they require the detection of the overload condition.

A Best-effort algorithm [viii] is based on the assumption that the probability of a high *value-density* task arriving is low. The value-density is defined by V/C, where V is the value of a task and C is its worst-case execution time. Given a set of tasks with defined values for successful completion, it can be shown that a sequence of tasks in decreasing order by value-density will produce the maximum value as compared to any other scheduling technique. The Best-effort algorithm admits tasks based on their value-densities and schedules them using the EDF policy. When higher value tasks are admitted, some lower value tasks may be deleted from the schedule or delayed until no other tasks with higher value exist. One key consideration in implementing such a policy is the estimation of current workload, which is either very difficult or very inaccurate in most practical systems that utilize WCET estimations. WCET estimation requires complex analysis of tasks [xi, xii], and, in most cases, the estimates are significantly larger than average execution times of tasks. Thus the Best-effort algorithms that use WCET to estimate loads may lead to sub-optimal value realization. Best-effort has been used as an overload control strategy for EDF.

Other approaches for detecting overload and rejecting tasks were reported in [xiii, xiv]. In the Guarantee scheme [xiii], the load on the processor is controlled by performing acceptance tests on new tasks entering the system. If the new task is found schedulable under worst-case assumptions, it is accepted; otherwise, the arriving task is rejected. In the Robust scheme [xiv], the acceptance test is based on EDF; if overloaded, one or more tasks may be rejected based on their importance. Because the Guarantee and Robust algorithms also rely on computing the schedules of tasks, often based on worst-case estimates, they usually lead to underutilization of resources. Thus Best-effort, Guarantee, or Robust scheduling algorithms are not good for soft real-time systems or applications that are generally referred to as "anytime" or "approximate" algorithms [x].

The combination of SJF and EDF, referred to as SCAN-EDF for disk scheduling, was proposed in [xv]. In the algorithm, SJF is only used to break a tie between tasks with identical deadlines. The work in [xvi, xvii] is very closely related to our idea of groups. This approach quantizes deadlines into deadline bins and places tasks into these bins. However, tasks within a bin (or group) are scheduled using *First Come First Served* (FCFS). The gEDF groups that we use are created dynamically instead of statically as done in [16,17].

One integrated real-time scheduler including Best-effort strategy for general-purpose operating systems has been proposed in [xviii]. However, this approach relies on the preemptive scheduling and uses Best-effort as an overload control strategy.

3. Real-time System Model

3.1 Definitions

A **job** τ_i in a real-time system or a **thread** in multithreading processing is defined as $\tau_i = (r_i, e_i, D_i, P_i)$; where r_i is its release time (or its arrival time); e_i is either its predicted worst-case or average execution time; D_i is its deadline. We also maintain a dynamic deadline $d_i = r_i + D_i$, which tracks the absolute time before the deadline expires. If modeling periodic jobs, P_i defines a task's periodicity. Note that aperiodic and sporadic jobs can be modeled by setting P_i appropriately.

For the experiments, we generated a fixed number (N) of jobs with varying arrivals, execution times and deadlines. We assume that the jobs are mutually independent. Each experiment terminated when the predetermined experimental time T expired. This permitted us to investigate the sensitivity of the various task parameters on the success rates of EDF and gEDF. We use random distributions available in MATLAB to generate the necessary parameters with tasks.

A **group** in the gEDF algorithm depends on a group range parameter Gr. τ_j belongs to the same group as τ_i if $d_i \le d_j \le (d_i + Gr*(d_i - t))$[2], where t is the current time, $1 \le i, j \le N$. In other words, we group jobs with very close deadlines together. We schedule groups based on EDF (all jobs in a group with an earlier deadline will be considered for scheduling before jobs in a group with later deadlines), but schedule jobs within a group using shortest job first (SJF) approach. Since SJF results in more (albeit shorter) jobs completing, intuitively gEDF should lead to a higher success rate than pure EDF.

We use the following notations for various parameters and computed values:

ρ: is the utilization of the system, $\rho = \Sigma e_i / T$. This is also called the load.

γ: is the success ratio, γ = the number of jobs completed successfully / N.

Tr: is the deadline tolerance for soft real-time systems. A job τ is schedulable if τ finishes before the time $(1 + Tr) * D$, where $Tr \ge 0$.

μ_e: is used either as the average execution time or the worst case execution time, and defines the expected value of the exponential distribution used for this purpose.

μ_r: is used to generate arrival times of jobs, and is the expected value of the exponential distribution used for this purpose.

μ_D: is the expected value of the random distribution used to generate task deadlines. We set this parameter as a multiple of μ_e.

$\Re$: is the average response time of the jobs.

∂: is the response-time ratio, $\partial = \Re / \mu_e$.

η_γ: is the success-ratio performance factor, $\eta_\gamma = \gamma_{gEDF} / \gamma_{EDF}$. This is used to compare gEDF with EDF.

η_∂: is the response-time performance factor, $\eta_\partial = \partial_{EDF} / \partial_{gEDF}$. This is used to compare gEDF with EDF.

3.2 gEDF Algorithm

We assume a uniprocessor system. Q_{gEDF} is a queue for gEDF scheduling. The current time is represented as t. $|Q|$ represents the length of the queue Q. $\tau = (r, e, D, P)$ yields the job at the head of the queue.

We define groups in gEDF as follows:

gEDF Group = $\{\tau_k \mid \tau_k \in Q_{gEDF}, d_k - d_1 \le D_1 * Gr, 1 \le k \le m$, where $m \le |Q_{gEDF}|\}$, where, D_1 is the deadline of the first job in a group.

Algorithm:

```
1.  Enqueue($Q_{gEDF}$, $\tau$)
      if ( $\tau$'s deadline $d > t$ )
          insert job $\tau$ into $Q_{gEDF}$ by Earliest Dead-
            lineFirst, i.e. $d_i \le d_{i+1} \le d_{i+2}$,
          where $\tau_i, \tau_{i+1}, \tau_{i+2} \in Q_{gEDF}$, $1 \le i \le |Q_{gEDF}| - 2$;
      end
2.  $\tau_{min}$ = Dequeue ($Q_{gEDF}$)
      if $Q_{gEDF} \ne \phi$
          find a job $\tau_{min}$ with $e_{min} = \min \{e_k \mid \tau_k \in Q_{gEDF}$,
          $d_k - d_1 \le Gr*D_1, 1 \le k \le m$, where
          $m \le |Q_{gEDF}|\}$;
          run it and delete $\tau_{min}$ from $Q_{gEDF}$;
      end
```

Enqueue is invoked on job arrivals and Dequeue is called when the processor becomes idle. The algorithm that we presented tends to favor smaller jobs and thus it does not always guarantee fairness. Also the algorithm needs to sort the jobs in each group, which could incur more overhead during execution than EDF. However, in most practical systems, the number of jobs in a group is small and the added runtime overhead will be negligible.

4. Numerical Results

MATLAB is used to generate tasks and the generated tasks are scheduled using EDF, gEDF, or other scheduling algorithms. For each chosen set of parameters, we have repeated each experiment 100 times (each time, generated N tasks using the random probability distributions and scheduled the generated tasks) and computed the average success rate. In what follows, we report the results and analyze the sensitivity of gEDF to the various parameters used in the experiments, the effects of the percentage of small jobs, and how well gEDF performs when compared to Best-Effort algorithm. Note that we use the non-preemptive task model.

[2] We are using the remaining time to a task deadline (called dynamic deadlines) in forming groups. We found that using static deadlines for defining groups did not significantly change the results.

4.1 Comparison of gEDF and EDF

4.1.1 Experiment 1 – Effect of Deadline Tolerance

Figures 1-3 show that gEDF achieves higher success rate than EDF when the deadline tolerance (i.e., soft real-time nature of the jobs) is varied from 20%, 50% to 100% (that is, a task can miss its deadline by 20%, 50% and 100%).

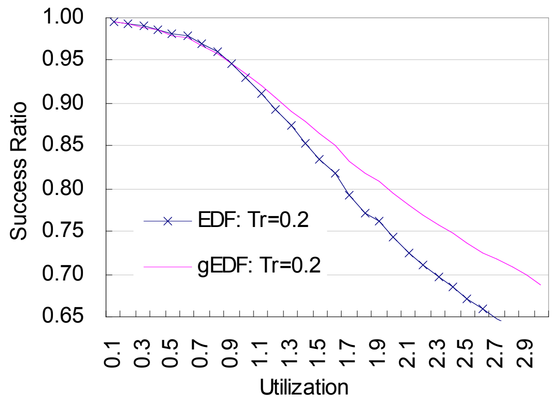

Figure 1: Success rates when deadline tolerance is 0.2.

For these experiments, we generated tasks by fixing expected execution rate and deadline parameters of the probability distributions, but varied arrival rate parameter to change the system load. The group range for these experiments is fixed at $Gr = 0.4$ (i.e., all jobs whose deadlines fall within 40% of the deadline of current job are in the same group). It should be noted that gEDF's success rates are consistently as good as those of EDF under light loads (utilization is less than 1), but higher than those of EDF under heavy loads (utilization is greater than 1, see the X-axis). Both EDF and gEDF achieve higher success rates when tasks are provided with greater deadline tolerance. The tolerance benefits gEDF more than EDF, particularly under heavy loads. Thus, gEDF is better suited for soft real-time tasks.

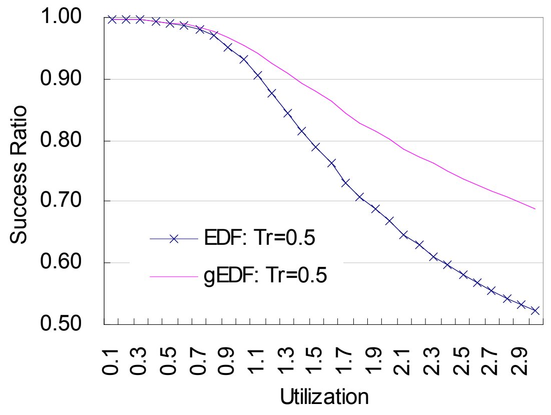

Figure 2: Success rates when deadline tolerance is 0.5.

Figure 4 summarizes these results by showing the percent improvement in success ratios achieved by gEDF when compared to EDF. The Y-axis shows that higher success rates are achieved by gEDF when compared to EDF for different system loads and different deadline tolerance parameters.

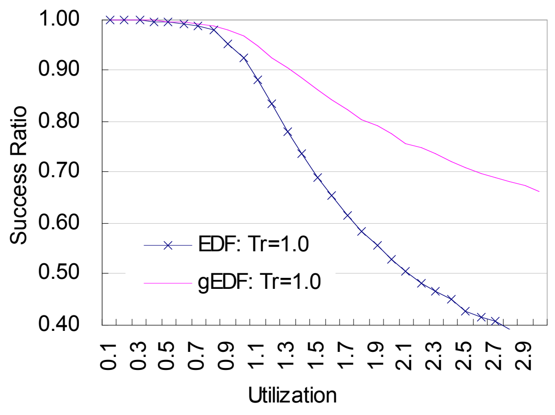

Figure 3: Success rates when deadline tolerance is 1.0.

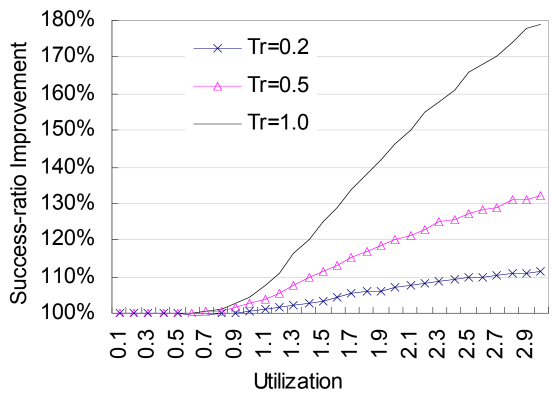

Figure 4: Success-ratio Performance Factor.

4.1.2 Experiment 2 - Effect of Deadline on Success Rates

In this experiment we explored the performance of EDF and gEDF when the deadlines are very tight (deadline = execution time) and when the deadlines are loose (deadline = 5 * execution time). Note that we generated the deadlines using exponential distribution with mean values set to 1 and 5 times the mean execution time μ_e. We varied the soft real-time parameter (*Tr*, or tolerance to deadline) in these experiments also, but all other parameters are kept the same as in the previous experiment. As can be seen in Figures 5 and 6, any scheduling algorithm will perform poorly for tight deadlines[3], except under ex-

[3] It should be noted that when $\mu_D = \mu_e$, all jobs should be scheduled immediately upon arrival, lest they misses their deadlines. The impact of using Least Laxity First

tremely light loads. Even under very tight deadlines, as in Figure 6, the deadline tolerance favors gEDF more than EDF. With looser deadlines, as in Figures 7 and 8, both EDF and gEDF achieve better performance. However, gEDF outperforms EDF consistently for all values of the deadline tolerance, *Tr*.

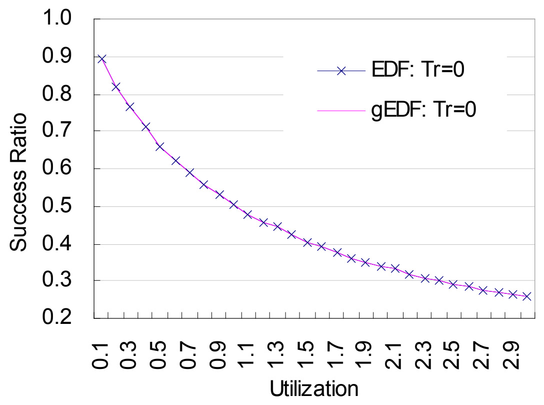

Figure 5: Tight deadline $\mu_D = 1$ (Deadline = Execution Time) and $Tr = 0$.

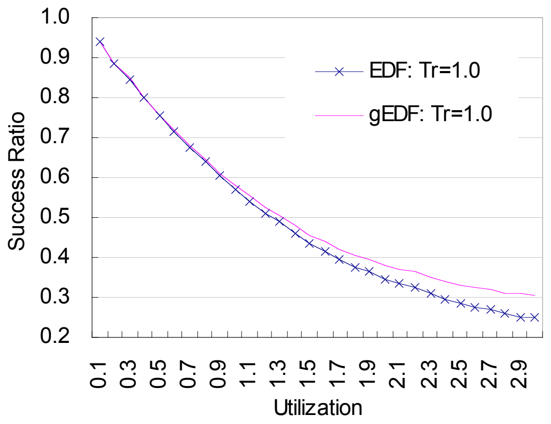

Figure 6: Tight deadline $\mu_D = 1$ (Deadline = Execution Time) and $Tr = 1.0$.

Figures 9 and 10 respectively, highlight the effect of deadlines on both EDF and gEDF. To more clearly evaluate how these approaches perform when the deadlines are very tight and loose, we set the deadlines to 1, 2, 5, 10 and 15 times the execution time of a task. We set $\mu_e = 40$, $Tr = 0.2$, (for gEDF $Gr = 0.4$). When $\mu_D = 1$ and 2, the success ratios of EDF and gEDF show no appreciable differences. However, when μ_D becomes reasonably large, such as 5, 10, and 15, the success ratio of gEDF is better than that of EDF.

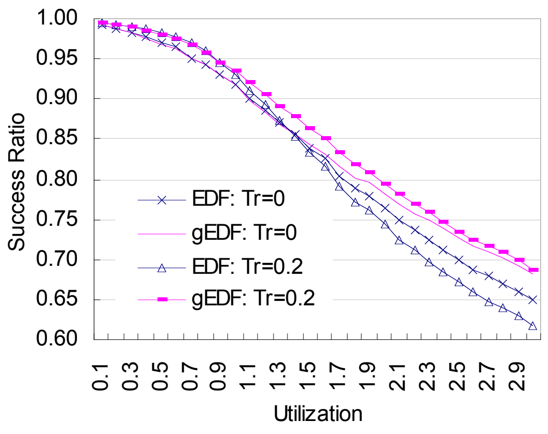

Figure 7: Looser deadline $\mu_D = 5$ (Deadline = 5* Execution Time) and $Tr = 0$ and 0.2.

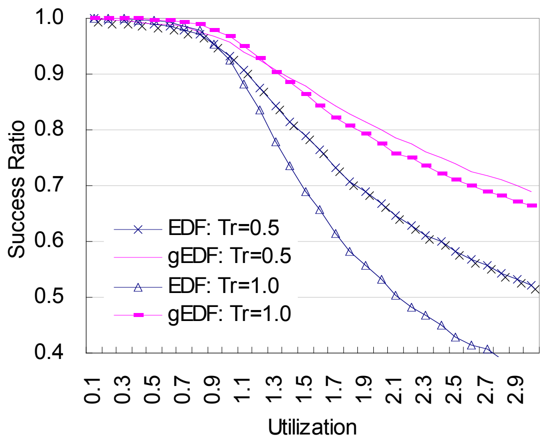

Figure 8: Looser deadline $\mu_D = 5$ (Deadline = 5* Execution Time) and $Tr = 0.5$ and 1.0.

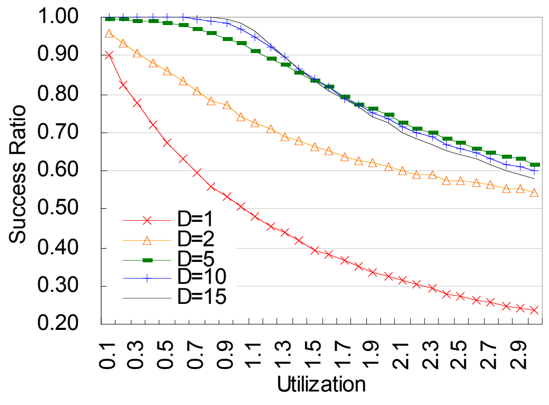

Figure 9: Success ratio of EDF when $\mu_D = 1$, 2, 5, 10, and 15.

approach is indirectly reflected by EDF when the deadlines are very tight.

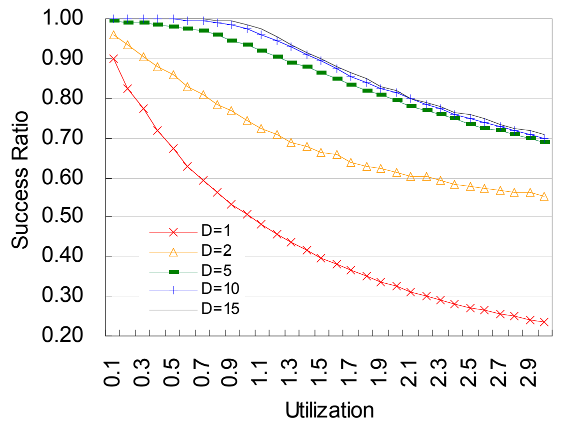

Figure 10: Success ratio of gEDF when μ_D = 1, 2, 5, 10, and 15.

Figure 11 summarizes these comparisons. The Y-axis shows the relative performance improvements (or better success ratios) achieved by gEDF over EDF.

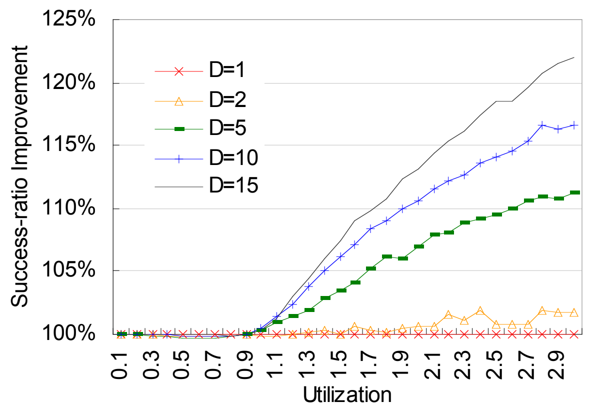

Figure 11: The ratio of success ratio of gEDF vs. success ratio of EDF when μ_D = 1, 2, 5, 10, and 15.

4.1.3 Experiment 3 - Effect of Group Range

In this experiment, we vary the group range parameter *Gr* for grouping tasks into a single group. Note in the following figures we do not include EDF data since EDF does not use groups. We set μ_D = 5 (Deadline = 5* Execution Time) and maintain the same values for other parameters as in the previous experiments. We set the deadline tolerance parameter *Tr* to 0.1 (10% tolerance in missing deadlines) in Figure 12, and to 0.5 (50% tolerance in missing deadlines) in Figure 13. The data shows that by increasing the size of a group, gEDF achieves higher success rates. In the limit, by setting the group range parameter to a large value, gEDF behaves more like SJF. There is a threshold value for the group size for achieving optimal success rate and the threshold depends on the execution time, tightness of deadlines and deadline tolerance parameters. For the experiments, we used a single exponential distribution for generating all task execution times. However, if we were to use a mix of tasks created using exponential distributions with different mean values, thus creating tasks with widely varying execution times, the group range parameter will have more pronounced effect on the success rates. Section 4.2 discusses the effect of different job classes, generated using different average execution time parameters.

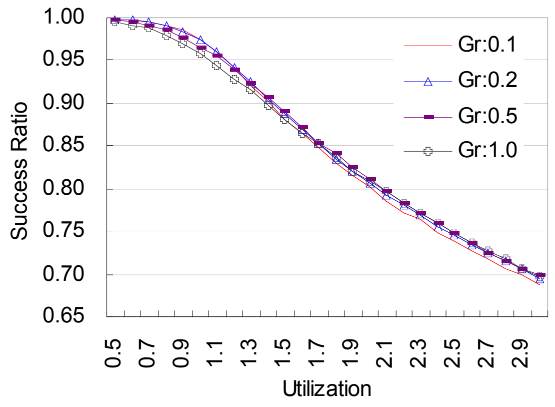

Figure 12: Group Range: *Gr* = 0.1, 0.2, 0.5, 1.0 (*Tr* = 0.1).

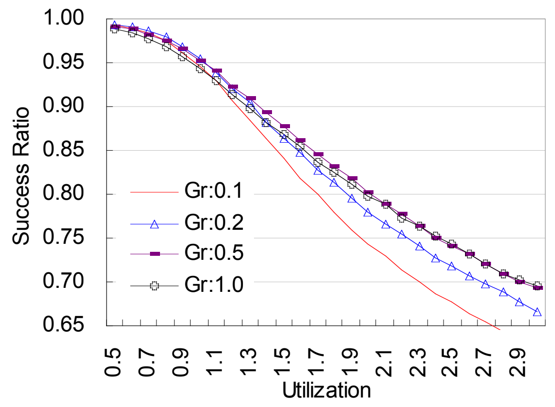

Figure 13: Group Range: *Gr* = 0.1, 0.2, 0.5, 1.0 (*Tr* = 0.5).

4.1.4 Experiment 4 – Effect of Deadline Tolerance on Response Time

Thus far we have shown that gEDF results in higher success rates than EDF, particularly when the system is overloaded. Next, we will compare the average response times achieved using gEDF with the response times achieved using EDF. Intuitively, completing shorter jobs first should result in faster response times. Our experiments support this. We set μ_e = 40, μ_D = 5, *Gr* = 0.4. Figures 14 and 15 show that gEDF can yield faster response times than EDF when soft real-time tolerance parameter *Tr* changes from 0 to 0.5, respectively.

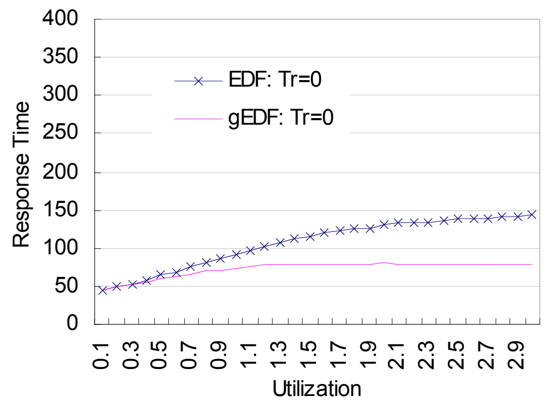

Figure 14: Response time when deadline tolerance $Tr = 0$.

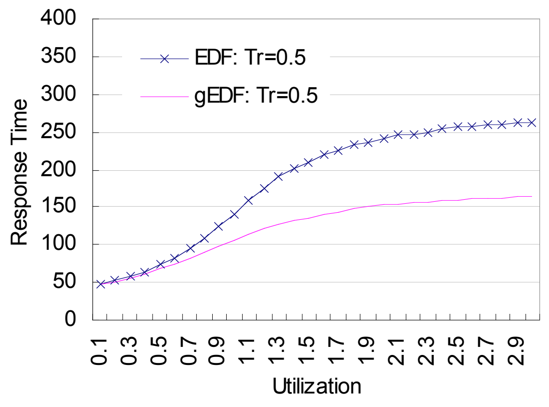

Figure 15: Response time when deadline tolerance $Tr = 0.5$.

Figure 16 summarizes the improvements in response times achieved by gEDF when compared to EDF. Note that that Y-axis shows the relative response times (and smaller number are better).

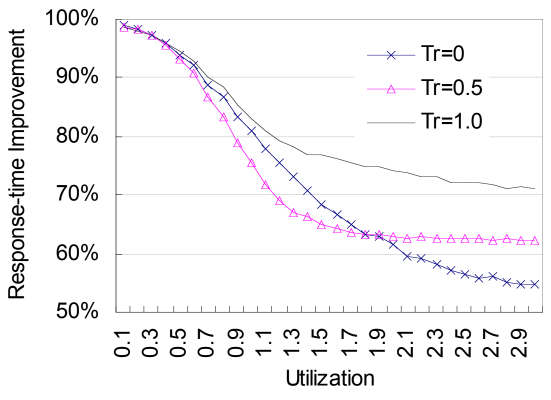

Figure 16: The ratio of response time of gEDF vs. response time of EDF.

4.1.5 Experiment 5 - The Effect of Tight Deadlines on Response Time

Figures 17 and 18 show the change in response time of EDF and gEDF when μ_D changes to 1, 2, 5, and 10. For these experiments, we set $\mu_r = \mu_e/\rho$, $\mu_e = 40$, $Gr = 0.4$, $Tr = 0.1$. Like the success ratios of EDF and gEDF, when μ_D is 1 and 2 times μ_e, there is no difference between EDF and gEDF. However, when μ_D is larger multiple of μ_e, gEDF results in faster response times.

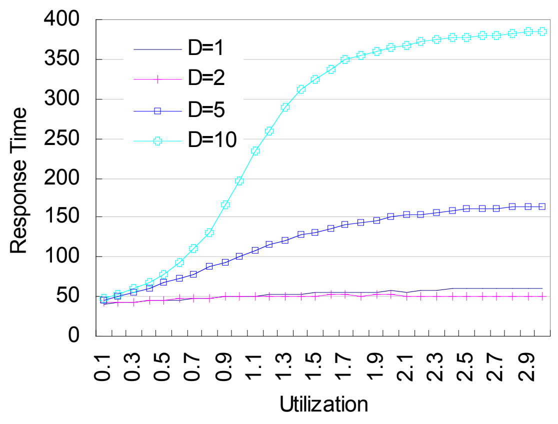

Figure 17: Response time of EDF when $\mu_D = 1, 2, 5$, and 10.

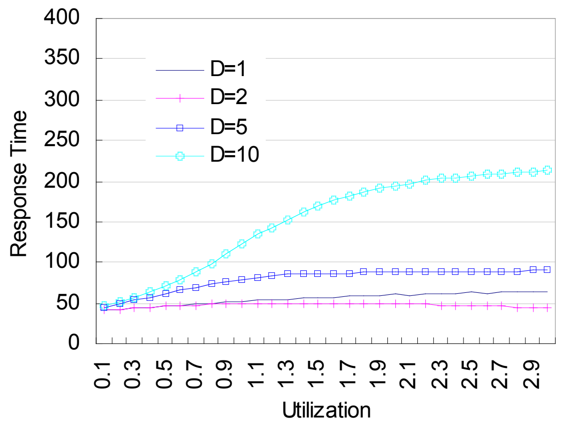

Figure 18: Response time of gEDF when $\mu_D = 1, 2, 5$, and 10.

Figure 19 summarizes the improvements in response times achieved by gEDF when compared to EDF. Note that that Y-axis shows the relative response times (and smaller number are better).

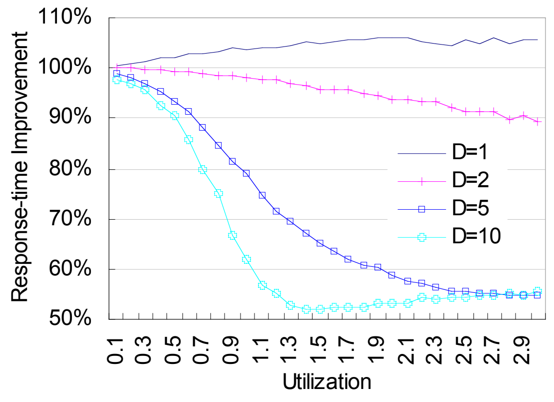

Figure 19: The ratio of response time of gEDF vs. response time of EDF when $\mu_D = 1, 2, 5$, and 10.

4.2 The Effect of Multiple Expected Execution Times

4.2.1 Experiment 6 – The Effect of Multiple μ_es on Success Ratio

The jobs generated in section 4.1 have a single average or worst case expected execution time μ_e. In other words, jobs were created using a single exponential distribution. To evaluate the impact of the case when jobs come from different classes with different mean execution times, we generated tasks using multiple exponential distributions with different mean values .

We use the following mean execution times for generating tasks. Note that a job class will be designated as (m, n) where m represent the mean value of the distribution used to generate execution times of tasks, and n represents the fraction of jobs (out of N) that are generated with the mean m.

Set-1: This is the base line consisting of jobs drawn from a single exponential distribution. We generate N jobs using an exponential distribution with a mean μ. We designate this set of jobs as ($\mu_{e,} N$).

Set-2: Here we have two types of jobs, one generated using a mean of (½)*$\mu_{e,}$ and the second with a mean of $\mu_{e,}$. Sixty-six percent of the jobs have a mean execution time of (½)$\mu_{e.}$ This set is designated by (1/2$\mu_{e,}$ 2/3N) and ($\mu_{e,}$ 1/3N).

Set-3: This set contains 3 classes of jobs generated using mean execution times of 1/4μ_e, 1/2$\mu_{e,}$ and $\mu_{e.}$ We designate this set as (1/4$\mu_{e,}$ 4/7N), (1/2$\mu_{e,}$ 2/7N), and ($\mu_{e,}$ 1/7N). Remember that the second number in each tuple represents the fraction of total number of jobs of each class.

Figure 20 shows that, when Tr is 0 (hard real-time), a job stream with more small jobs do not improve the success ratios. On the other hand, when dealing with soft real-time jobs (with a deadline tolerance Tr of 0.2 and 0.5), job classes do impact success ratios of gEDF as shown in Figures 21 and 22. Note that Set 2 and Set 3 have larger number of smaller jobs than Set 1. As expected gEDF results in higher success rates over EDF when there are more small jobs.

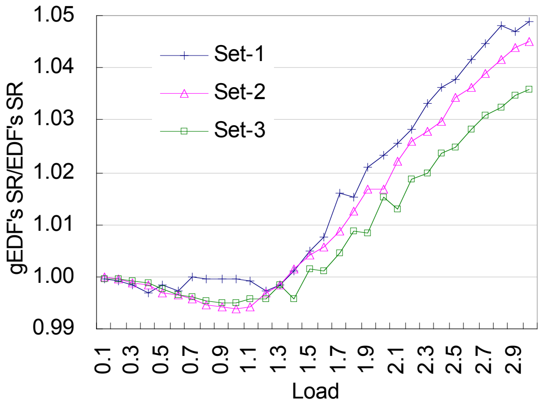

Figure 20: Success ratio of gEDF/success ratio of EDF when $Tr = 0$.

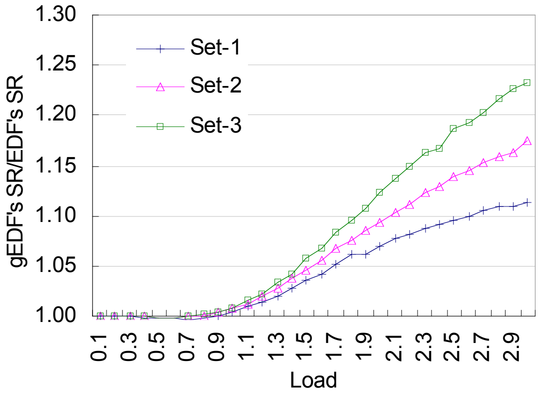

Figure 21: Success Ratio of gEDF/success ratio of EDF when $Tr = 0.2$.

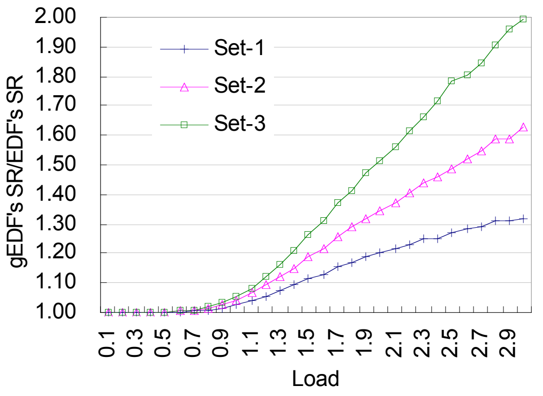

Figure 22: Success Ratio of gEDF/success ratio of EDF when $Tr = 0.5$.

4.2.2 Experiment 7 – The Effect of Percentage of Small Jobs on Success Ratio

Previously, we analyzed the effect of data sets with different job classes using different μ_es, and observed that a workload with more small jobs show higher gEDF performance when compared to EDF. In this section, we will analyze the case where we use two different job classes (with two different μ_es) but change the percentage of small jobs in the mix.

Distribution 1: all jobs with μ_e.
Distribution 2: 1/2jobs with μ_e; 1/2 jobs with $1/2\mu_e$.
Distribution 3: 2/5jobs with μ_e; 3/5jobs with 1/3 μ_e.
Distribution 4: 1/5 jobs with μ_e; 4/5 jobs with 1/8 μ_e.

We set Tr = 0.5. Figure 23 shows that the distribution with more small jobs, gEDF obtains higher success ratios than EDF. Note the Distribution 4 has more small jobs than any other distribution, and the data shows that gEDF benefits from this fact.

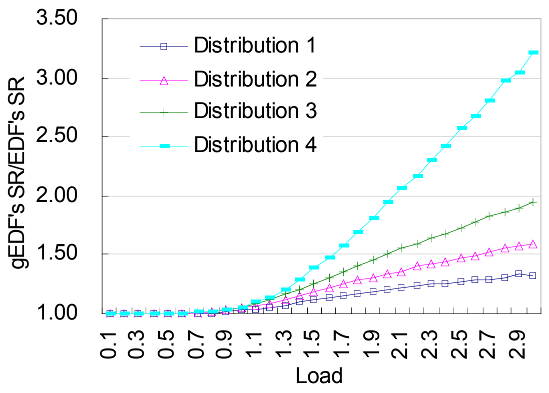

Figure 23: Success ratio of gEDF/success ratio of EDF Tr = 0.5.

4.3 Comparisons of gEDF, Best-effort, and Guarantee Algorithms

4.3.1 Experiment 8 - Comparison of Success Ratios of gEDF and Best-effort

We have shown that gEDF not only shows better performance than EDF under overloaded conditions, but shows comparable or better performance than EDF when the system is underloaded. Thus, there is no need to switch between EDF and gEDF based on system load. Researchers have explored adaptive algorithms to control the performance when the system is overloaded. One such algorithm is called the Best-effort Algorithm (see Section 2). In this paper we will use the same best effort criteria (i.e., value-density: V/C) that Locke [8] used. For this experiment we set all jobs to have the same value. The Best-effort approach used EDF when the system is under-loaded, and attempts to maximize V/C when the system exceeds 100% utilization (i.e., overloaded conditions).

The Best-effort relies on the precise estimation or prediction of utilization for switching between EDF algorithm and the Best-effort. While it may be possible to predict the system load when the system only processes periodic jobs, it is very difficult to compute the system load if the system processes a mixture of periodic, aperiodic, and sporadic jobs. Recently, synthetic utilization bound has been proposed to measure real utilization. For the EDF-based schemes, however, synthetic utilization and real utilization are very close [xix]. The estimated loads are imprecise because most real-time systems utilize worst-case execution times (WCET), and in most cases the actual utilization of the system is lower than these estimates. Switching to Best-Effort based on such imprecise load estimations leads to inefficient utilization of the resources. In this paper we use a clairvoyant scheme based on actual execution times of the real-time jobs. Thus the comparisons shown here are present the most optimistic scenarios as far as the Best-Effort algorithm is concerned.

We set $\mu_r = \mu_e/\rho$, $\mu_e = 20$, $\mu_D = 5$, $Gr = 0.4$. Figures 24 and 25 show that gEDF achieves higher success rates than Best-effort when the deadline tolerance is varied, Tr = 0.2, 0.5, and 1.0.

Considering the need for predicting the precise utilization for implementing Best-Effort, any improvements gained by gEDF should be viewed in a positive light. The performance gains achieved by gEDF are even grater when the deadline tolerance is as lenient as 50%, as in Figure 25 (even for lighter loads).

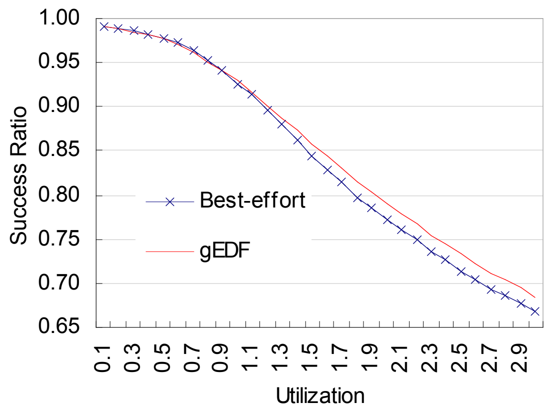

Figure 24: Success rates when deadline tolerance is 0.2.

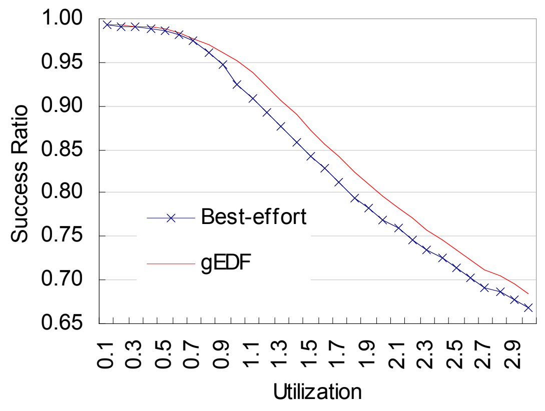

Figure 25: Success rates when deadline tolerance is 0.5.

4.3.2 Experiment 9 – Comparison of Response Times of gEDF and Best-effort

Figures 26 and 27 compare the average response times achieved using gEDF with that achieved using Best-effort. We set $\mu_r = \mu_e/\rho$, $\mu_e = 20$, $\mu_D = 5$, $Gr = 0.4$.

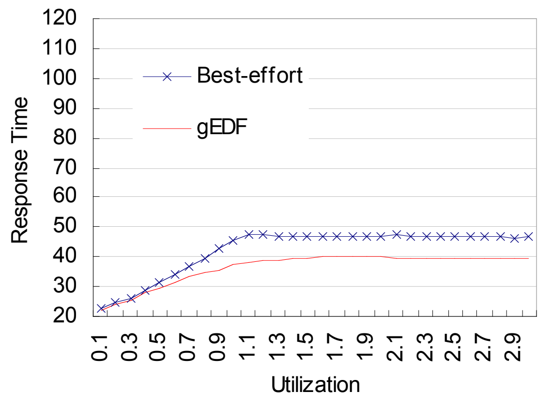

Figure 26: Response time when deadline tolerance is 0.

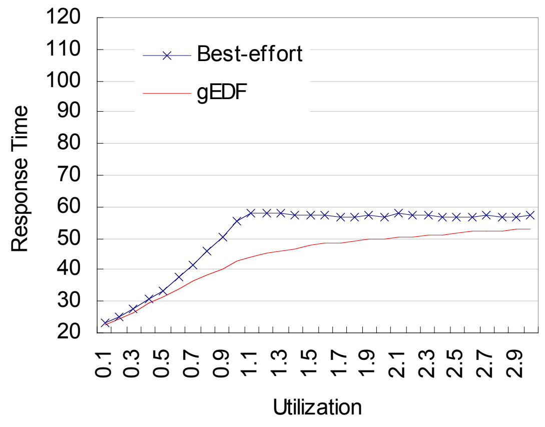

Figure 27: Response time when deadline tolerance is 0.2.

4.3.3 Experiment 10 – Comparison of Success Ratios of gEDF and Guarantee

Although Guarantee algorithm is inappropriate for soft real-time systems, we include a comparison of gEDF with the Guarantee scheme here for the sake of completeness. When the system is underloaded, Guarantee uses EDF; when the system is overloaded, Guarantee uses a specific policy to choose real-time jobs and guarantees execution of the jobs by their deadlines. In the simulation used here, incoming jobs, are accepted based on FCFS policy, if they can be scheduled (along with all jobs already guaranteed) by the deadline.

We set $\mu_r = \mu_e/\rho$, $\mu_e = 20$, $\mu_D = 5$, $Gr = 0.4$. Figures 28 and 29 show the success ratios of all the real-time scheduling algorithms discussed in this paper, including the Guarantee algorithm, Best-effort, EDF, and gEDF.

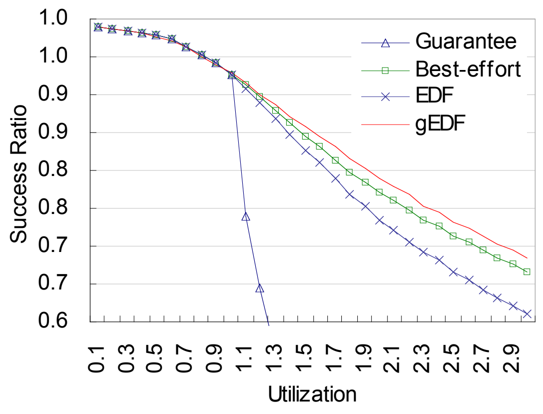

Figure 28: Success ratio when deadline tolerance is 0.2.

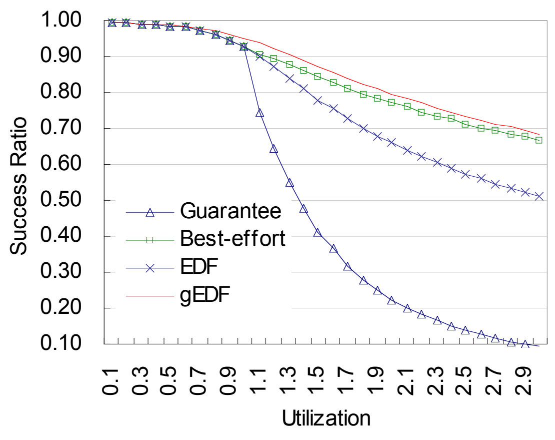

Figure 29: Success ratio when deadline tolerance is 0.5.

4.3.4 Experiment 11 – Comparison of the Response Times of gEDF and Guarantee

We set $\mu_r = \mu_e/\rho$, $\mu_e = 20$, $\mu_D = 5$, $Gr = 0.4$. Figure 30 compares the response times of the real-time algorithms considered in this paper.

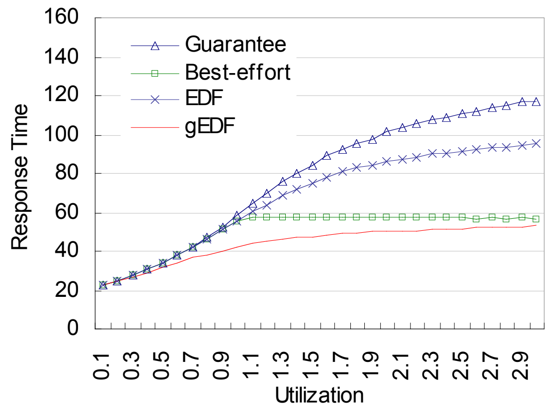

Figure 30: Response times when deadline tolerance is 0.2.

5. Conclusions and Future Work

In this paper, we presented a new real-time scheduling algorithm that combines Shortest Job First scheduling with the Earliest Deadline First scheduling. We grouped tasks with deadlines that are very close to each other, and scheduled jobs within a group using SJF scheduling. We have shown that group EDF results in higher success rates (that is, the number of jobs that have completed successfully before their deadline) as well as in faster response times.

It has been known that while EDF produces an optimum schedule (if one is available) for systems using preemptive scheduling, EDF is not as widely used for non-preemptive systems. We believe that for soft real-time systems that utilize multithreaded processors, non-preemptive scheduling is more efficient. Although EDF produces practically acceptable performance even for non-preemptive systems when the system is underloaded, EDF performs very poorly when the system is heavily loaded. Our gEDF algorithm performs as well as EDF in terms of success ratio when a system is underloaded. Even on systems that are underloaded, gEDF shows higher success rates than EDF when dealing with soft real-time tasks (using higher deadline tolerances). And gEDF consistently outperforms EDF in overloaded situations.

In this paper we also compared our gEDF with schemes that adapt EDF when the system is overloaded. Among the adaptive algorithms, we considered the Best-Effort and Guarantee algorithms. In general, gEDF, which can be used in both overloaded and under-loaded situations, performs as well as or better than EDF, Best-Effort and Guarantee schemes. It should be remembered the last two adaptive algorithms require the ability to accurately measure system loads so that the overloaded conditions can be detected. In most cases this is very difficult, particularly if the workload consists of periodic, aperiodic and sporadic jobs, or if the system consists of both real-time and non-real-time jobs. Moreover, estimating system load based on worst-case execution times, leads to under-utilizations, thus predicting overloaded conditions incorrectly. These problems are not encountered by gEDF, since there is no need to estimate system load or to switch between EDF and Best-Effort on overloads.

In future work, we plan to explore the impact of a variety of parameters on the performance gEDF, and evaluate gEDF for real workloads.

6. References

[i] F. Balarin, L. Lavagno, P. Murthy, and A. S. Vincentelli, “Scheduling for Embedded Real-Time Systems”, IEEE Design & Test of Computer, January-March, 1998.

[ii] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”, Journal of the ACM, Vol. 20. No. 1, pp. 46-61.

[iii] R. Jain, C. J. Hughes, and S. V. Adve, “Soft Real-Time Scheduling on Simultaneous Multithreaded Processors”, In Proceedings of the 23rd IEEE International Real-Time Systems Symposium, December 2002.

[iv] K. M. Kavi, R. Giorgi, and J. Arul, “Scheduled Dataflow: Execution Paradigm, Architecture, and Performance Evaluation”, IEEE Transactions on Computers, Vol. 50, No. 8, August 2001.

[v] K. Jeffay and C. U. Martel, “On Non-Preemptive Scheduling of Periodic and Sporadic Tasks”, Proceedings of the 12th IEEE Real-Time Systems Symposium, San Antonio, Texas, December 1991, IEEE Computer Society Press, pp. 129-139.

[vi] M. R. Garey, D. S. Johnson, “Computer and Intractability, a Guide to the Theory of NP-Completeness”, W. H. Freeman Company, San Francisco, 1979.

[vii] L. Georges, P. Muehlethaler, N. Rivierre, “A Few Results on Non-Preemptive Real-time Scheduling”, INRIA Research Report nRR3926, 2000.

[viii] C. D. Locke, “Best-effort Decision Making for Real-Time Scheduling”, CMU-CS-86-134 (PhD Thesis), Computer Science Department, Carnegie-Mellon University, 1986.

[ix] J. K. Dey, J. Kurose, and D. Towsley, “Online Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks”, Tech. Rep. 93-09, Department of Computer Science, University of Massachusetts, Amherst, Jan 1993.

[x] S. Zilberstein, “Using Anytime Algorithms in Intelligent Systems”, AI Magazine, fall 1996, pp.71-83.

[xi] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm, “The Influence of Processor Architecture on the Design and the Results of WCET Tools”, Proceedings of IEEE July 2003, Special Issue on Real-time Systems.

[xii] G. Bernat, A. Collin, and S. M. Petters, “WCET Analysis of Probabilistic Hard Real-Time Systems”, IEEE Real-Time Systems Symposium 2002, 279-288.

[xiii] G. Buttazzo, M. Spuri, and F. Sensini, Scuola Normale Superiore, Pisa, Italy, “Value vs. Deadline Scheduling in Overload Conditions”, 16th IEEE Real-Time Systems Symposium (RTSS’95) December 05-07, 1995.

[xiv] S. K. Baruah and J. R. Haritsa, “Scheduling for Overload in Real-Time Systems”, IEEE Transactions on Computers, Vol. 46, No. 9, September 1997.

[xv] A. L. N. Reddy and J. Wyllie, “Disk Scheduling in Multimedia I/O system”, In Proceedings of ACM multimedia’93, Anaheim, CA, 225-234, August 1993.

[xvi] B. D. Doytchinov, J. P. Lehoczky, and S. E. Shreve, “Real-Time Queues in Heavy Traffic with Earliest-Deadline-First Queue Discipline”, Annals of Applied Probability, No. 11, 2001.

[xvii] J. P. Hansen, H. Zhu, J. P. Lehoczky, and R. Rajkumar, “Quantized EDF Scheduling in a Stochastic Environment”, Proceedings of the International Parallel and Distributed Processing Symposium, 2002.

[xviii] Jason Nieh, and Monica S. Lam, “A SMART Scheduler for Multimedia Applications”, ACM Transactions on Computer Systems, Vol. 21, No. 2, May 2003.

[xix] T. Abdelzaher, V. Sharma, and C. Lu, “A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling”, IEEE Trans. On Computers, March 2004.

Wenming Li received BS in computer science from Sichuan University, China, MS in computer engineering from Institute of Computing Technology of Chinese Academy of Sciences, and MS in computer science from University of North Texas in 1985, 1990, and 2001 respectively. He was a senior researcher in Chinese Academy of Sciences and a computer engineer in Atmel Corporation and Tarrant Appraisal District. He is currently a PhD candidate in computer science in University of North Texas. His current research interests are computer architecture, real-time, and embedded systems.

Krishna Kavi is currently a professor and the Chairman of Computer Science and Engineering department at the University of North Texas. Previously he held the Eminent Scholar Chair professorship at the University of Alabama in Huntsville, and a professorship at the University of Texas at Arlington. He was a Scientific Program Director at the US National Science Foundation between 1993-1995.

His research interests are primarily in the various aspects of Computer Architecture. He also conducted research on formal methods for the design and verification of software systems, agent-based formalisms, performance and reliability analyses of computer systems using Petri nets. He authored or co-authored more than 150 technical publications.

Robert Akl received the B.S. degree in computer science from Washington University in St. Louis, in 1994, and the B.S., M.S. and D.Sc. degrees in electrical engineering in 1994, 1996, and 2000, respectively. He also received the Dual Degree Engineering Outstanding Senior Award from Washington University in 1993.

Dr. Akl is currently an Assistant Professor at the University of North Texas, Department of Computer Science and Engineering. In 2002, he was an Assistant Professor at the University of New Orleans, Department of Electrical and Computer Engineering. From October 2000 to December 2001, he was a senior systems engineer at Comspace Corporation, Coppell, TX. His research interests include wireless communication and network design and optimization.