An Efficient Non-Preemptive Real-Time Scheduling

Wenming Li, Krishna Kavi and Robert Akl
Department of Computer Science and Engineering
The University of North Texas
Denton, Texas 76203, USA
{wenming, kavi, rakl}@cse.unt.edu

Abstract

Traditional real-time systems are 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. Our goal in this research is to improve the success rate of the well-known Earliest Deadline First (EDF) approach even when the load on the system is very high. 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 and EDF using randomly generated tasks with varying execution times, release times, deadlines and tolerance to missing deadlines, under varying workloads. We believe that the grouping of tasks with similar deadlines and utilizing information other than deadlines (such as execution times, priorities or resource availability) for scheduling tasks within a group can lead to new and more efficient real-time scheduling algorithms.

Keywords: Real-time scheduling, Earliest Deadline First (EDF), Shortest Job First (SJF), Best-Effort Scheduling, Approximate Computations.

1. Introduction

The *Earliest Deadline First* (EDF) algorithm is the most widely used scheduling algorithm for real-time systems [1]. For a set of preemptive tasks (be they periodic, aperiodic, or sporadic), EDF will find a schedule if a schedule is possible [2]. 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) [3, 4]. The application of EDF for non-preemptive tasks is not as widely studied. 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 [5, 6, 7]. 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 EDF approach leads to dramatically poor performance (low success rates) [8]. 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 – this is 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 in a more aggressive scheduling 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 analyzed a variation of EDF that can improve the success ratio (that is, the number of tasks that have been successfully scheduled to meet their deadlines), particularly in overloads. It 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 each other), 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 both under overload and under-load conditions. The computational complexity of gEDF is the same as that of EDF. In this paper, we evaluate the performance of gEDF using randomly generated tasks with varying execution times, release times, deadlines 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 [9, 10]. 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.

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 used as a dynamic priority-driven scheduling scheme [5]. EDF has a much higher efficiency than many other scheduling algorithms including static *Rate-Monotonic* scheduling algorithm. For preemptive tasks, EDF is able to reach up to the maximum possible processor utilization when lightly loaded. Although finding an optimal schedule for periodic and aperiodic non-preemptive tasks is NP-hard [6, 7], our experiments show 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 require the detection of the overload condition.

A Best-effort algorithm [8] is proposed 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 successfully completing, it can be shown that a sequence of tasks in decreasing order by value-density will produce maximum value as compared to any other scheduling technique. The Best-effort algorithm admits tasks for scheduling based upon 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 for implementation of 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 [11, 12], and in most cases, they are significantly greater than average execution times of tasks. Thus the Best-effort algorithm that use WCET to estimate loads may lead to sub-optimal value realization.

Other approaches for detecting overload and rejecting tasks were reported in [13, 14]. In the Guarantee scheme [13], 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 assumption, it is accepted; otherwise, the arriving task is rejected. In the Robust scheme [14], 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 lead to poor utilization of resources. Best-effort, Guarantee, or Robust scheduling algorithms, which utilize worst case execution times, and hard deadlines are not good for soft real-time systems or applications that are generally referred to as "anytime" or "approximate" algorithms [10].

The combination of SJF and EDF, referred to as SCAN-EDF for disk scheduling, was proposed in [15]. In the algorithm, SJF is only used to break a tie between tasks with identical deadlines. The work in [16, 17] is very closely related to our ideas of groups. They quantize deadlines into deadline bins and place tasks into these bins. However, tasks within a bin (or group) are scheduled using FIFO.

3. Real-time System Model

A **job** τ_i in real-time systems or a **thread** in multi-threading processing is defined as $\tau_i = (r_i, e_i, D_i, P_i)$; r_i is its release time (or the time when it arrives in the system); e_i is either its predicted worst-case or average execution time; D_i is its deadline. If modeling periodic jobs, P_i defines a task's periodicity. Note that aperiodic jobs can be modeled by setting P_i to infinity, sporadic tasks by setting P_i to a variable. For the experiments, we generated a fixed number (N) of jobs. Each experiment terminated when the experimental time T expired. This permitted us to investigate the sensitivity of the various task parameters on the success rates of EDF and gEDF. Most of the parameters are based on exponential distributions available in MATLAB.

A **group** in the gEDF algorithm depends on a group range parameter Gr. τ_j belongs to the same group as τ_i if $D_i \le D_j \le (D_i + Gr*D_i)$, where $1 \le i, j \le N$. In other words, we group jobs with very close deadlines. 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 SJF. Since SJF results in more (albeit shorter) jobs completing, intuitively gEDF should lead to 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 and acceptable 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_t: 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.

$\Re$: 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, $\eta_\gamma = \gamma_{gEDF} / \gamma_{EDF}$.

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

4. Numerical Results

MATLAB was used to generate tasks and schedule the tasks using both EDF and gEDF. For each chosen set of parameters, we repeated each experiment 100 times (generated N tasks using the random probability distributions and scheduled the generated tasks) and computed average success rates. In what follows, we report the results and analyze the sensitivity of gEDF to the various parameters used in the experiments. Note that we use the non-preemptive task model.

4.1 Experiment 1 – Effect of Deadline Tolerance

Figures 1-3 show that gEDF achieves higher success rate than EDF, when the deadline tolerance is varied to 20%, 50% and 100% (that is, a task can miss its deadline by 20%, 50% and 100%).

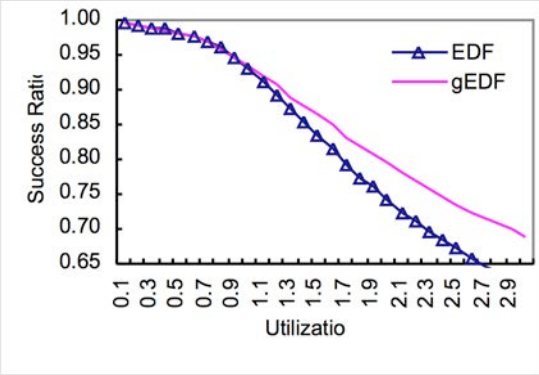

Figure 1: Success rates when deadline tolerance is 0.2.

For these experiments, we generated tasks by fixing expected execution rate and deadline parameters of the probability distributions, but varied arrival rate parameters to change the system load. The group range for these experiments is fixed at *Gr* = 0.4. It should be noted that gEDF consistently performs as well as EDF under light loads (utilization is less than 1) but outperforms EDF under heavy loads (utilization is greater than 1). 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.

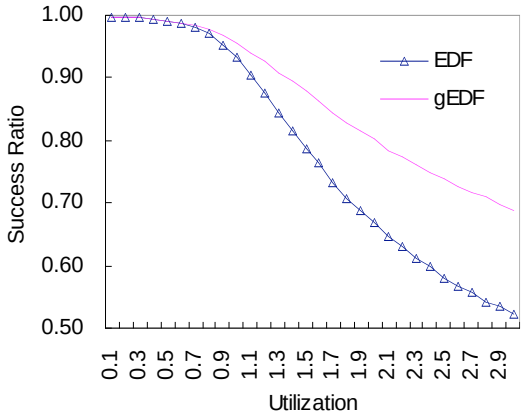

Figure 2: Success rates when deadline tolerance is 0.5.

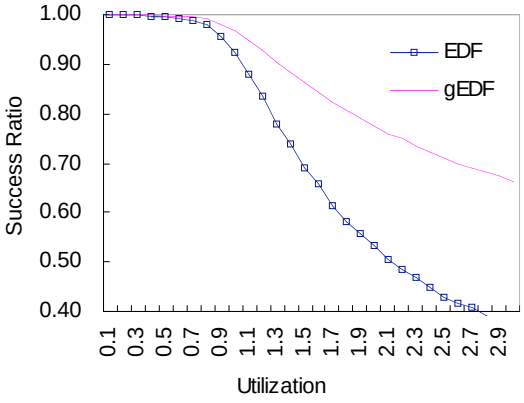

Figure 3: Success rates when deadline tolerance is 1.0.

Figure 4 summarizes these results by showing the percent improvement in success ratios achieved by gEDF as compared to EDF.

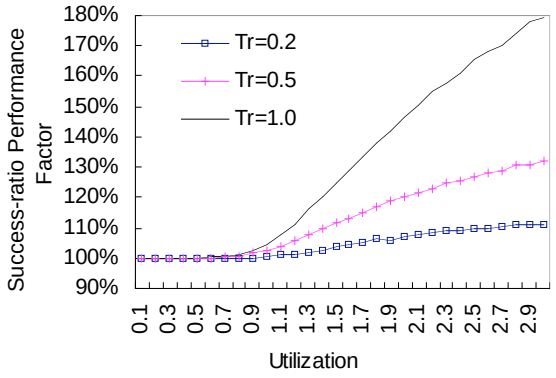

Figure 4: Success-ratio Performance Factor.

4.2 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. The other parameters were kept the same as the previous experiment. As can be seen in Figures 5 and 6 any scheduling algorithm will perform poorly for tight deadlines, except under extremely 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*.

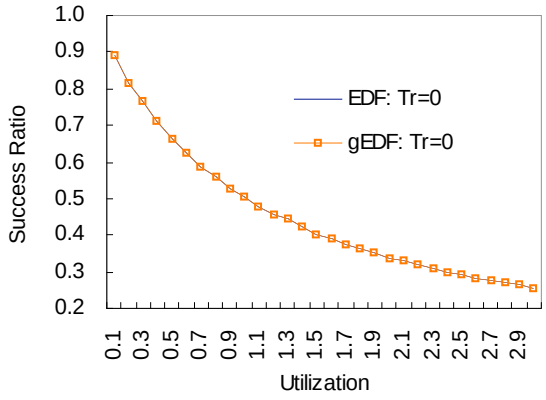

Figure 5: Tight deadline μ_D = 1 (Deadline = Execution Time) and *Tr* = 0.

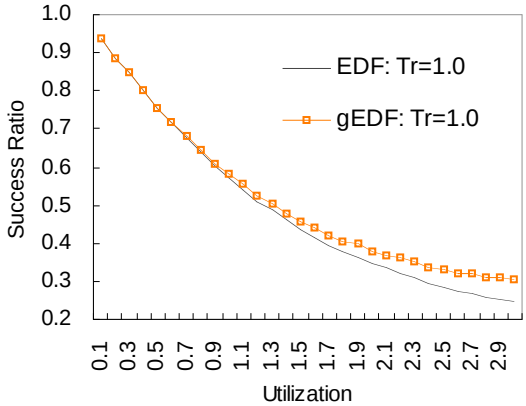

Figure 6: Tight deadline μ_D = 1 (Deadline = Execution Time) and *Tr* = 1.0.

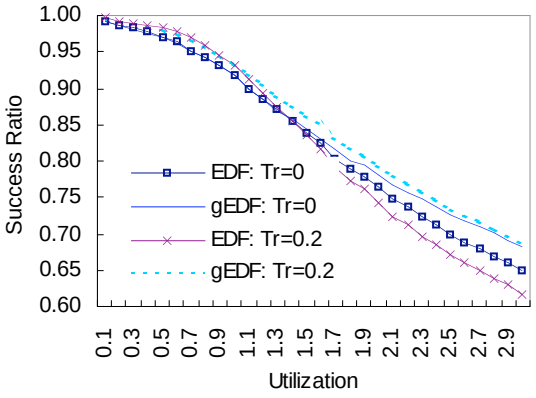

Figure 7: Looser deadline μ_D = 5 (Deadline = 5* Execution Time) and *Tr* = 0 and 0.2.

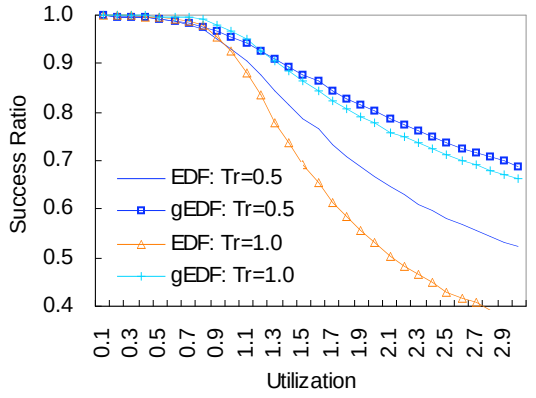

Figure 8: Looser deadline μ_D = 5 (Deadline = 5* Execution Time) and *Tr* = 0.5 and 1.0.

Figures 9 and 10 highlight the effect of deadlines on both EDF and gEDF, respectively.

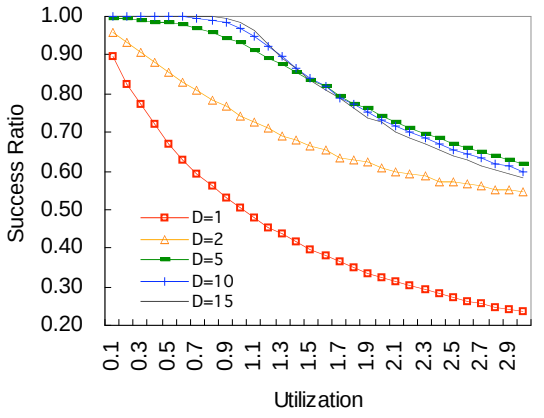

Figure 9: Success ratio of EDF when μ_D = 1, 2, 5, 10, and 15.

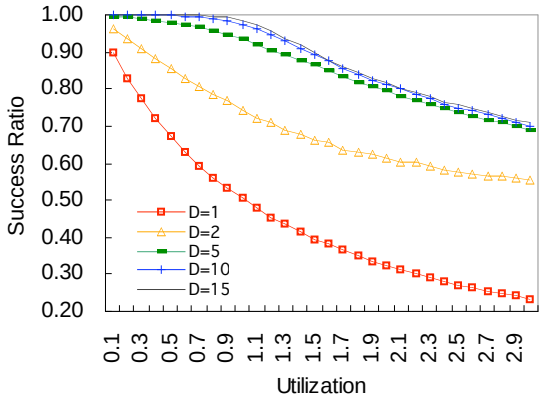

Figure 10: Success ratio of gEDF when μ_D = 1, 2, 5, 10, 15.

To more clearly evaluate how these approaches perform when the deadlines are very tight and loose, we set the deadlines to be 1, 2, 5, 10 and 15 times the execution time of a task. We set $\mu_r = \mu_e/\rho$, μ_e = 40, *Tr* = 0.2, (for gEDF *Gr* = 0.4). When μ_D = 1 and 2, the success ratios of EDF and gEDF have no apparent difference. However, when μ_D becomes reasonably large, such as 5, 10, and 15, the success ratio of gEDF is better than that of EDF.

4.3 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 11, and to 0.5 (50% tolerance in missing deadlines) in Figure 12.

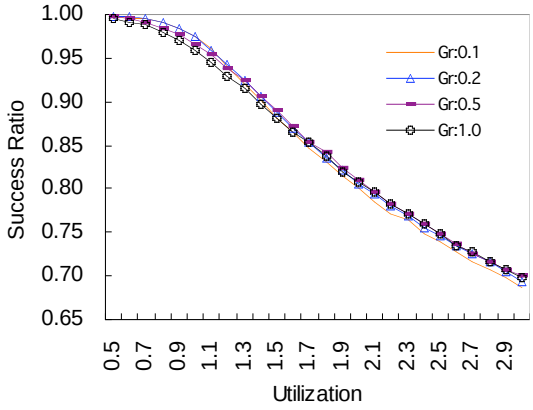

Figure 10: Group Range: *Gr* = 0.1, 0.2, 0.5, 1.0 (*Tr* = 0.1).

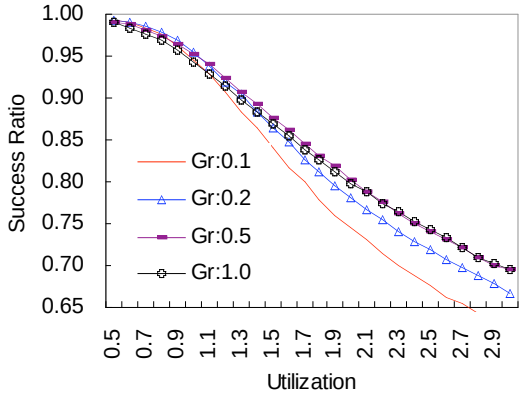

Figure 11: Group Range: *Gr* = 0.1, 0.2, 0.5, 1.0 (*Tr* = 0.5).

The data shows that by increasing the size of a group, gEDF achieves higher success rate. 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, deadline 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.

4.4 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 EDF. We set $\mu_r = \mu_e/\rho$, μ_e = 40, μ_D = 5, *Gr* = 0.4.

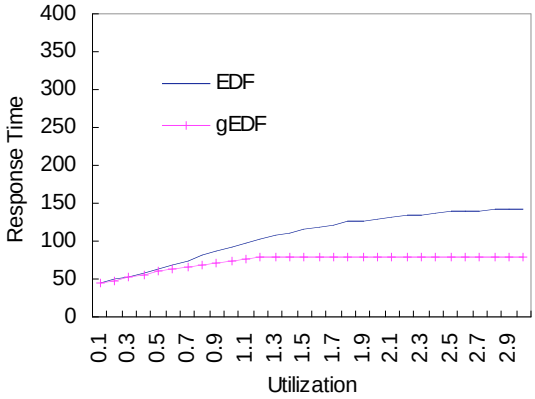

Figure 13: Response time when deadline tolerance *Tr* = 0.

Figures 13, 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 to 1.0, respectively.

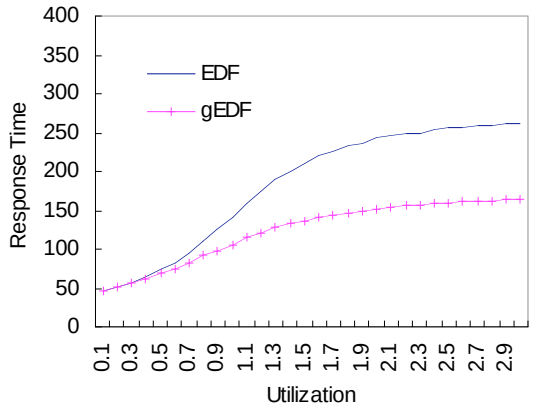

Figure 14: Response time when deadline tolerance *Tr* = 0.5.

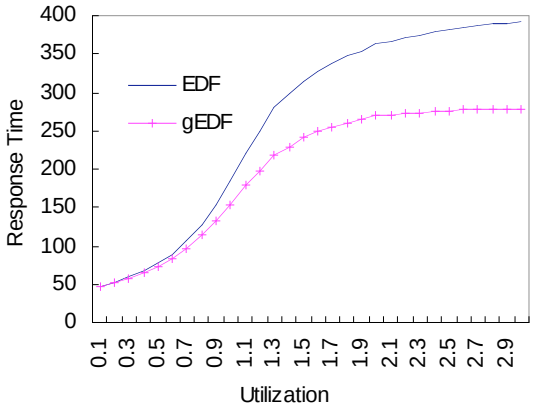

Figure 15: Response time when deadline tolerance *Tr* = 1.0.

4.5 Experiment 5 - The Effect of Deadline on Response Time

We set $\mu_r = \mu_e/\rho$, $\mu_e = 40$, $Gr = 0.4$, $Tr = 0.1$. Figures 16 and 17 show the change in response time of EDF and gEDF when μ_D changes to 1, 2, 5, and 10. Like the success ratios of EDF and gEDF, when μ_D is very small such as 1 and 2, there is no difference between EDF and gEDF. However, when μ_D is larger then gEDF results in faster response times.

5. Conclusions and Future Work

In this paper, we presented a new real-time scheduling algorithm that combines Shortest Job First scheduling with Earliest Deadline First scheduling. We grouped tasks that have 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.

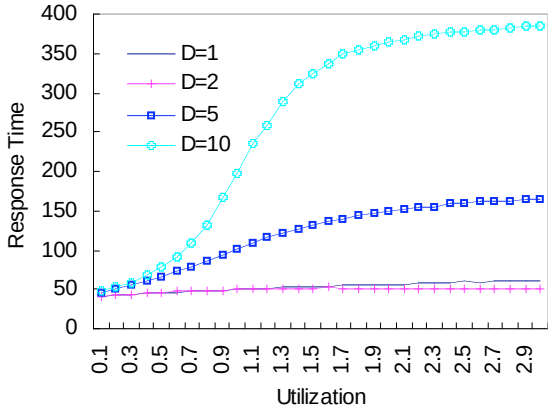

Figure 16: Response time of EDF when μ_D = 1, 2, 5, and 10.

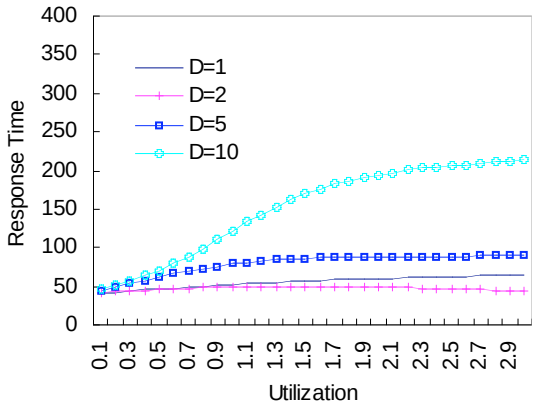

Figure 17: Response time of gEDF when μ_D = 1, 2, 5, and 10.

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 under-loaded, EDF performs very poorly when the system is heavily loaded. Our gEDF algorithm performs as well as EDF when a system is under-loaded, but outperforms EDF when the system is overloaded. gEDF also performs better than EDF for soft real-time systems, where a task is allowed to miss its deadline.

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

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

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