Guaranteed Dynamic Priority Assignment Schemes for Real-Time Tasks with (*m*, *k*)-Firm Deadlines

Hyeonjoong Cho, Yongwha Chung, and Daihee Park

We present guaranteed dynamic priority assignment schemes for multiple real-time tasks subject to (*m*, *k*)-firm deadlines. The proposed schemes have two scheduling objectives: providing a bounded probability of missing (*m*, *k*)-firm constraints and maximizing the probability of deadline satisfactions. The second scheduling objective is especially necessary in order to provide the best quality of service as well as to satisfy the minimum requirements expressed by (*m*, *k*)-firm deadlines. We analytically establish that the proposed schemes provide a guarantee on the bounded probability of missing (*m*, *k*)-firm constraints. Experimental studies validate our analytical results and confirm the effectiveness and superiority of the proposed schemes with regard to their scheduling objectives.

Keywords: Real-time systems, (*m*, *k*)-firm deadline, guaranteed dynamic priority assignment.

I. Introduction

Traditionally, real-time systems are classified into two categories: hard and soft. For hard real-time systems, the deadline of each real-time task must be met, whereas for soft real-time systems, occasional deadline-misses are acceptable. Hard real-time applications, including aircraft control, robotics, and automated manufacturing, risk catastrophe unless the deadlines of all activities are completely satisfied. In contrast, multimedia service, an example of soft real-time systems, allows occasional deadline-misses that do not cause significant performance degradation.

For soft real-time systems, one of the classic representations for the tolerance of deadline-misses is the maximum allowable deadline-miss ratio. For instance, a data stream could be specified to allow a 10% deadline-miss ratio. However, such a statistical representation cannot precisely specify the distribution of deadline-misses during a given time period, as pointed out in [1]. In other words, a statistical representation cannot distinguish between evenly-distributed deadline-misses and consecutive deadline-misses. Even though they have the same 10% deadline-miss ratio, a difference in the distribution of deadline-misses results in a significant difference in the quality of service. Besides, even for hard real-time systems, some level of deadline-misses can be tolerated, as described in [2]. For example, a control system can have a few occasional deadline misses without a significant degradation in control performance, provided that there are only a limited number of consecutive deadline-misses.

To precisely state such time constraints, Hamdaoui and others defined the (*m*, *k*)-firm deadline as $0<m\leq k$. The (*m*, *k*)-firm deadline expresses the acceptable quality of service, where at least *m* instances of a task in any window of *k*

Manuscript received Sept. 16, 2009; revised Jan. 14, 2010; accepted Feb. 6, 2010.

This research was financially supported by the Ministry of Education, Science Technology (MEST) and Korea Industrial Technology Foundation (KOTEF), Rep. of Korea, through the Human Resource Training Project for Regional Innovation.

Hyeonjoong Cho (corresponding author, phone: +82 41 860 1374, email: raycho@korea.ac.kr), Yongwha Chung (email: ychungy@korea.ac.kr), and Daihee Park (email: dhpark@korea.ac.kr) are with the Department of Computer and Information Science, Korea University, Sejong, Rep. of Korea.

doi:10.4218/etrij.10.0109.0544

consecutive instances meet their deadlines [1]. As an example, although both the (9, 10)-firm deadline and the (90, 100)-firm deadline statistically imply a similar maximum deadline-miss ratio of 10%, the distributions of their deadline misses are quite different. A task that violates its own (m, k)-firm deadline, that is, when there are fewer than m deadline satisfactions occurring in a window of k consecutive instances, introduces a *dynamic failure*. Thus, the occurrence rate of dynamic failures can be used as a metric to measure how often the quality of service falls below the required level.

Several attempts have been made to deal with the problem of scheduling multiple real-time tasks constrained by (m, k)-firm deadlines. These are classified into static and dynamic approaches. A static approach follows a predetermined sequence of skippable invocations of tasks. Ramanathan and others proposed a static approach that partitions tasks' invocations into mandatory and optional, where mandatory invocations must meet their deadlines in order to satisfy the (m, k)-firm deadline constraint, whereas optional invocations are skippable [3]. On the other hand, dynamic approaches make scheduling decisions on which invocations of tasks should be executed at runtime. Hamdaoui and others proposed a dynamic approach, the distance-based priority (DBP) scheme, which schedules tasks based on the recent history of the system [1]. More specifically, DBP assigns a priority value equal to the minimum number of consecutive misses required for the task to fall into a dynamic failure state, where a task with lower priority values has higher priority. Thus, a higher priority is given to a task having a shorter distance to its dynamic failure state. In this paper we focus on dynamic rather than static approaches in order to consider less restricted environments that do not require off-line predetermination about task invocations.

DBP is a dynamic scheme to minimize the probability of dynamic failures, that is, it is a best-effort technique. As indicated in [2], DBP has two limitations: first, there is no schedulability test to ensure that there is no probability, or even bounded probability, of dynamic failures and second, only homogeneous task sets with the same execution time, inter-arrival time, etc. are considered. In addition, DBP does not consider maximizing the quality of service even when it is possible. For example, the objective of real-time multimedia stream scheduling is not only to provide guaranteed performance by bounding the probability of dynamic failures of streams as constrained by (m, k)-firm deadlines, but also to provide the best quality of service, that is to maximize deadline satisfaction of streams.

To address these issues, we propose the guaranteed dynamic priority assignment (GDPA) scheme that schedules multiple real-time tasks constrained by (m, k)-firm deadlines. GDPA is designed to provide a bounded probability of dynamic failures for multiple tasks with (m, k)-firm deadlines when the system is under-loaded and maximize the probability of deadline satisfactions. Interestingly, GDPA is a generic real-time scheduling algorithm that provides scheduling optimality for (1,1)-firm constrained tasks, that is, regular hard real-time tasks, in the sense that GDPA satisfies all task deadlines when the system is under-loaded, as does an optimal real-time scheduling algorithm, earliest deadline first (EDF) [4].

The rest of the paper is organized as follows. Section II briefly reviews several related works. Section III describes our activity models. Section IV discusses a specific example in order to show our motivations. Section V provides detailed descriptions on the proposed algorithms. Section VI reports our simulation-based experimental studies. We conclude in section VII.

II. Related Works

Hamdaoui and others defined the (m, k)-firm deadline and proposed a dynamic priority assignment technique, DBP [1]. [5] presented an analytical model for computing the probability of dynamic failures on DBP. As discussed, DBP is an on-line, best-effort scheduling technique based on the recent history of the system. Bernet and others introduced the notion of weakly-hard real-time constraints as a generalization of (m, k)-firm deadlines [6]. Weakly-hard real-time constraints allow much richer types of temporal requirements, including (m, k)-firm deadlines. In [7], Chen and others pointed that prior works use per-stream state information for scheduling, which incurs computational and spatial overhead and eventually restricts scalability. Instead, they proposed the class selection algorithm (CSA) that adjusts the trade-off between scalability and quality of service (QoS) granularity. CSA adopts the integrated DBP (IDBP) for its scheduling algorithm [8]. IDBP is an improved DBP that considers not only the distance to dynamic failure but also the distance to exit out of a failure state. In [9], Matrix-DBP was proposed to improve DBP by considering not only local information, that is, distance to failure, but also richer information such as periods, service times, and its relationship with other streams. Matrix-DBP was extended to address task models other than periodic ones in [10]. These schemes proposed in [9] and [10] are non-preemptive scheduling, whereas we are considering preemptive scheduling algorithms in this paper.

Several static and dynamic approaches for (m, k)-firm deadlines have been proposed. Ramanathan and others proposed a static approach that partitions the invocations of

tasks into mandatory and optional [3]. Quan and others reduced interferences between mandatory jobs and improved the partition scheme [11]. As a dynamic approach, Bernet and others proposed a guaranteed, on-line scheme: a bi-modal scheduler (BMS) [2]. BMS provides tasks with two modes: normal and panic. Tasks are first scheduled in normal mode, and a task that is likely to cause a dynamic failure falls into panic mode. BMS assigns fixed priorities to tasks in panic mode, which are higher than those of tasks in normal mode. This allows interference only between tasks in panic mode, and therefore BMS provides a guarantee on its performance by providing a schedulability test for tasks in panic mode. Since BMS utilizes a fixed priority scheduling strategy, it inherits the properties of fixed priority scheduling strategies including low least upper bound of processor utilization [12]. Here we focus on dynamic priority scheduling strategies for job-level priority changes rather than fixed ones for task-level priority changes.

To maximize the quality of service and satisfy the requirements of (m, k)-firm deadlines, Lin and others defined a new metric, a granularity of quality of service reward (GQoS-reward), and a heuristic for static approaches to maximize the total GQoS-reward [13].

There have been many attempts to apply (m, k)-firm deadline constraints to various applications, including a vehicle control system [14], a plant control [15], and real-time streams on multihop networks [16]. Recently, researchers have been addressing more complicated problems combining (m, k)-firm deadline constraints and other research issues including energy-efficiency, multiprocessors, etc [17], [6], [18], and [19].

III. Activity Model

We consider an application that consists of a set of tasks, denoted $\{T_1, T_2, \cdots, T_n\}$, where n is the number of tasks. Each T_i has a number of invocations, that is, jobs, which are released periodically or sporadically with known inter-arrival times. The j-th job of T_i is denoted by $J_{i,j}$. The inter-arrival time of T_i is denoted by p_i and the worst-case execution time of T_i is denoted by c_i. Each T_i has a (m_i, k_i)-firm deadline constraint ($0<m_i \leq k_i$). Thus, each T_i is characterized by (p_i, c_i, m_i, k_i). We assume that each task's relative deadline is the same as its period for the sake of simplicity, but we emphasize that GDPA is also designed to support tasks with different relative deadlines and periods. We also assume that none of the tasks share any resources nor any precedence.

As defined in [1], let $\delta_{i,j}$ denote the status of the j-th job of the i-th task, where $\delta_{i,j}$ is a *miss* (denoted by m) or a *meet* (denoted by M) of its deadline. The state of a task at a given time can be represented as k_i-tuple ($\delta_{i,j-k+1}$,$\cdots$, $\delta_{i,j-1}$, $\delta_{i,j}$), where j is the index of the most recent job of T_i that meets or misses the deadline. If T_i is in a state where it has less than m_i meets out of k_i-tuple, this is defined as a dynamic failure state.

Suppose that T_i has a (1, 3)-firm deadline and is in state (MmM), where (MmM) denotes that the most recent job met, the previous one missed, and the one before that met each deadline. Let $l_i(x,state)$ denote the position (from right) of the x-th meet in $state$ of T_i. Then, $l_i(1,(\text{MmM})) = 1$. The distance, $Dist_i$ is defined as $k_i - l_i(m_i,state)+1$ which is the minimum number of consecutive deadline-misses for T_i to fall into a dynamic failure state. In the above-mentioned example, $Dist_i=3$ and a dynamic failure will occur if three consecutive deadline-misses follow. When T_i is in the dynamic failure state, $Dist_i$ becomes zero.

IV. Motivations

DBP assigns a high priority to a task T_i having a low $Dist_i$, that is, it has a high probability of falling into a dynamic failure state. This best-effort scheme considerably restrains the occurrence of dynamic failures but it does not maximize the quality of service even when it is possible.

Suppose three tasks are constrained by (5, 3, 2, 4), (14, 2, 1, 2), and (26, 6, 2, 3). As shown in Fig. 1, the jobs of the three tasks are released simultaneously at time zero. Assume that each task has a distance, 3, 2, and 2, respectively, at time zero. This is equivalent to the assumption that all previous jobs of the three tasks before time zero have met their deadlines.

When the total utilization demand, defined as $\sum c_i/p_i$, is less than or equal to one, that is, the system is under-loaded, EDF satisfies all deadlines. In this example, the total utilization demand is 0.97, and thus, EDF can provide 100% deadline satisfaction. On the other hand, DBP misses T_1's deadline at time 5 (we assume that DBP selects a task with the earliest

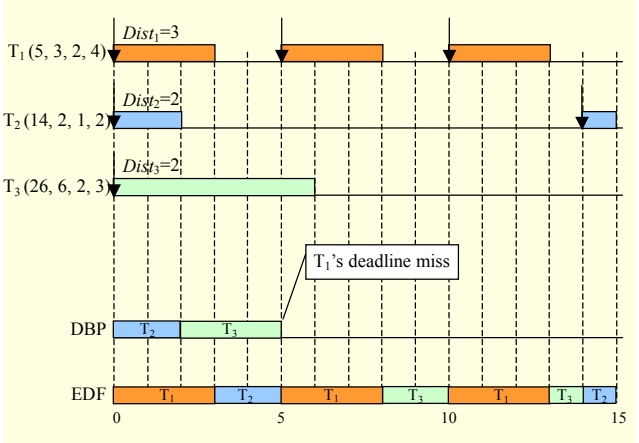


Fig. 1. Comparison between EDF and DBP.

deadline when several tasks have the same distance, as suggested in [1]). DBP increases the chance of meeting deadlines by assigning high priority to a task close to its dynamic failure state. This strategy can be improved by taking into account both deadlines and distances of tasks for scheduling decisions, especially when the system is under-loaded. In Fig. 1, for instance, although T_1 has the longest distance, selecting T_1 first for execution makes the whole task set feasible, since T_1 has the earliest deadline. When there are no deadline-misses, there are no dynamic failures. This was the motivation for the design of a guaranteed on-line scheduling algorithm, GDPA.

V. Algorithms

A high-level description of GDPA is shown in algorithm 1. GDPA is invoked at two events, task arrival and task completion. We define the following functions to describe GDPA.

The isJobFeasible($J_{i,k}$) function returns a boolean value denoting a job $J_{i,k}$'s feasibility, that is, whether or not a job meets its deadline without considering its correlation with other jobs-for task abortion. We consider three different policies for job abortion, including no abortion, normal abortion, and antecedent abortion. The no abortion policy allows no abortion. Normal abortion aborts a job that has already missed its deadline. Antecedent abortion aggressively aborts a job that has a high probability of missing its deadline, that is, the predicted completion time each job exceeds its deadline. The body of this function is based on the selection of such an abortion policy. For example, this function returns true when the job's deadline is greater than or equal to the current time in the normal abortion policy. On the other hand, in the antecedent abortion policy, it returns true when the difference between the job's deadline and the current time is greater than or equal to the job's remaining execution time.

- The computeDistance($J_{i,k}$) function computes each task's distance $Dist_i$ to its dynamic failure state.
- The sortByShortestDistanceFirst(ζ) function sorts jobs of ζ in order of the shortest distance first.
- The insertByEarliestDeadlineFirst($J_{i,k}$, σ) function inserts a job into σ in order of the earliest deadline first.
- The isFeasible(σ) function returns a boolean value denoting a schedule σ's feasibility−that is, whether or not the whole set of jobs in the queue meet their deadlines.
- The remove($J_{i,k}$, σ) function removes a job $J_{i,k}$ from schedule σ.
- The headOf(σ) function returns a job that is at the head of schedule σ.

Algorithm 1. High-level description of GDPA.

```
1:  Input:        Ready queue ζ that contains tasks in ready state
2:  Output:       A selected task for execution
3:  Variables:    Temporal queue σ1, σ2=φ
4:
5:  for all J_{i,k} ∈ ζ do
6:      if ! isJobFeasible (J_{i,k}) do
7:              abort (J_{i,k});
8:      end if
9:  end for
10: for all J_{i,k} ∈ ζ do
11:     computeDistance (J_{i,k});
12: end for
13: σ1=sortByShortestDistanceFirst (ζ);
14: for all J_{i,k} ∈ σ1 from head to tail do
15:     insertByEarliestDeadlineFirst(J_{i,k}, σ2);
16:     if ! isFeasible(σ2) do
17:             remove (J_{i,k}, σ2);
18:     end if
19: end for
20: return headOf(σ2);
```

In lines 5 to 9, GDPA aborts jobs which are not feasible based on selection of a job abortion policy. In line 11, the computeDistance() function calculates each job's distance to its dynamic failure state. In line 15, GDPA inserts the job closest to its dynamic failure state first into queue σ_2 in order of the earliest deadline first. Whenever GDPA inserts a job, it checks if the constructed job sequence is feasible and, if not, it removes that job. Finally, GDPA selects a task at the head of σ_2 containing jobs sorted in the EDF order. In this manner, GDPA considers both distances and deadlines in order to reduce the probability of dynamic failures while maximizing the probability of deadline satisfaction.

Theorem 1. Suppose that a set of independent periodic tasks is running in the absence of CPU overloads, and sufficient cycles exist for meeting all task deadlines, that is, $\sum c_i/p_i \leq 1$ ($0<i\leq n$). Then, a schedule produced by EDF is also produced by GDPA, and there is no dynamic failure.

Proof. Consider algorithm 1. For periodic tasks in the under-loaded situation, σ_2 in line 20 is deadline-ordered. As proved in [4], a deadline-ordered schedule is optimal with respect to meeting all deadlines when there is no over-load. Thus, GDPA results in no deadline-misses as with EDF, and no dynamic failure, by definition. □

GDPA's computation cost depends on the functions and a loop from lines 13 to 19. With n number of tasks, the sortByShortestDistanceFirst() function costs $O(n\log n)$. Both insertByEarliestDeadlineFirst() and isFeasible() functions cost $O(n)$. There is a maximum of n iterations, and thus, the entire cost of GDPA approaches $O(n^2)$. Although the computational cost of GDPA is a little higher than that of EDF and DBP, it is

justifiable, especially for multimedia streaming applications where the inter-arrival times of frames (or jobs) are on the order of milliseconds.

In spite of this argument, a low-cost version of GDPA, simplified GDPA (GDPA-S), is presented in algorithm 2 for applications other than multimedia streaming.

Algorithm 2. High-level description of GDPA-S.

```
1:  Input: Ready queue ζ that contains tasks in ready state
2:  Output: A selected task for execution
3:  Static variables: Queue σ_edf , σ_sdf
4:
5:  switch triggering event do
6:  case task_release(J_i,k):
7:          insertByEarliestDeadlineFirst(J_i,k , σ_edf );
8:          insertByShortestDistanceFirst(J_i,k , σ_sdf );
9:  case task_complete(J_i,k):
10:         remove(J_i,k , σ_edf );
11:         remove(J_i,k , σ_sdf );
12:         computeDistance(T_i);
13:  end switch
14:  for  all  J_i,k ∈ ζ do
15:      if  ! isJobFeasible (J_i,k)  do
16:              abort (J_i,k);
17:      end if
18:  end for
19:  if isFeasible(σ_edf ) then
20:         return headOf(σ_edf );
21:  else
22:         return headOf(σ_sdf);
23:  end if
```

Two static queues, σ_{edf} and σ_{sdf}, are maintained from the time the system starts. Both queues are updated at two scheduling events: task release and task completion. The distance of T_1 is updated only at a task completion event. The cost from line 6 to 13 is $O(\log n)$. From line 19 to 23, when σ_{edf} is feasible, GDPA-S selects the earliest deadline job for execution, otherwise it selects the shortest deadline job. The cost from line 19 to 23 is $O(n)$, which becomes the computational cost of the entire GDPA-S. In the over-loaded situation, we assume that GDPA-S utilizes the shortest remaining-execution-time job when several jobs have the same distance: we assume this because when two jobs have the same probability of falling into the dynamic failure state, utilizing the shortest remaining-execution-time job can save CPU cycles. Note that GDPA-S has the same property as in theorem 1, which is omitted for simplicity.

An example of EDF and DBP behavior in the under-loaded situation are illustrated in Fig. 1. Note that both GDPA and GDPA-S are designed to behave like EDF in the under-loaded situation and thus, the schedule by EDF in Fig. 1 is exactly the same as those by GDPA and GDPA-S. In the over-loaded situation, on the other hand, GDPA and GDPA-S behave differently. Suppose that three tasks are constrained by (5, 3, 2, 4), (14, 2, 1, 2), and (26, 13, 2, 3), which is a similar situation to the previous example except that T_3 has a longer execution time. In such over-loaded cases, DBP selects the T_2 having the shortest distance to its dynamic failure state and EDF selects the T_1 having the earliest deadline. However, GDPA that selected the T_1 in the under-loaded situation makes more deliberate scheduling decision in this case. It first sorts tasks in order, with the shortest distance first, that is, T_2, T_3, T_1, and inserts those tasks into the EDF queue. When it inserts the third task, T_1, this causes infeasibility and T_1 is removed. Finally, GDPA selects the task at the head of EDF queue, T_2. In summary, GDPA makes the same scheduling decision as EDF in the under-loaded situation, and it makes a smart scheduling decision to reduce the probability of dynamic failure in the over-loaded situation.

VI. Experimental Evaluation

In order to validate the above-mentioned features and evaluate the performance, simulation-based experimental studies were conducted. EDF and DBP were selected as the two counterparts to GDPA and GDPA-S. EDF is a well-known optimal real-time scheduling algorithm for tasks constrained by deadlines, and DBP is the first on-line dynamic scheduling algorithm for tasks constrained by (m, k)-firm deadlines. DBP is assumed to select the earliest deadline job when several jobs have the same distance. For all schemes, we applied two different job abortion policies: normal and antecedent job abortion. The no-abortion policy was not considered, since it drastically degrades the performance of both EDF and DBP due to the domino effect [20]. Note that GDPA and GDPA-S are domino-effect-free.

We considered two cases. In the first case, all tasks are constrained by (1, 1)-firm deadlines, that is, regular hard real-time constraints. In the second case, each task T_i is constrained by a (m_i, k_i)-firm deadline. The purpose of the two experiments was to evaluate the generic performances of GDPA and GDPA-S with respect to the other methods.

Two metrics were used to measure the performance: probability of deadline satisfaction (PDS) and probability of dynamic failures (PDF). PDS is defined to be the ratio of deadline satisfactions to task releases, while PDF is defined to be the ratio of dynamic failures to task releases. PDS represents how effectively a scheduling algorithm provides quality of service, while PDF represents how effectively a scheduling algorithm satisfies the minimum timeliness requirement expressed with (m, k)-firm deadlines.

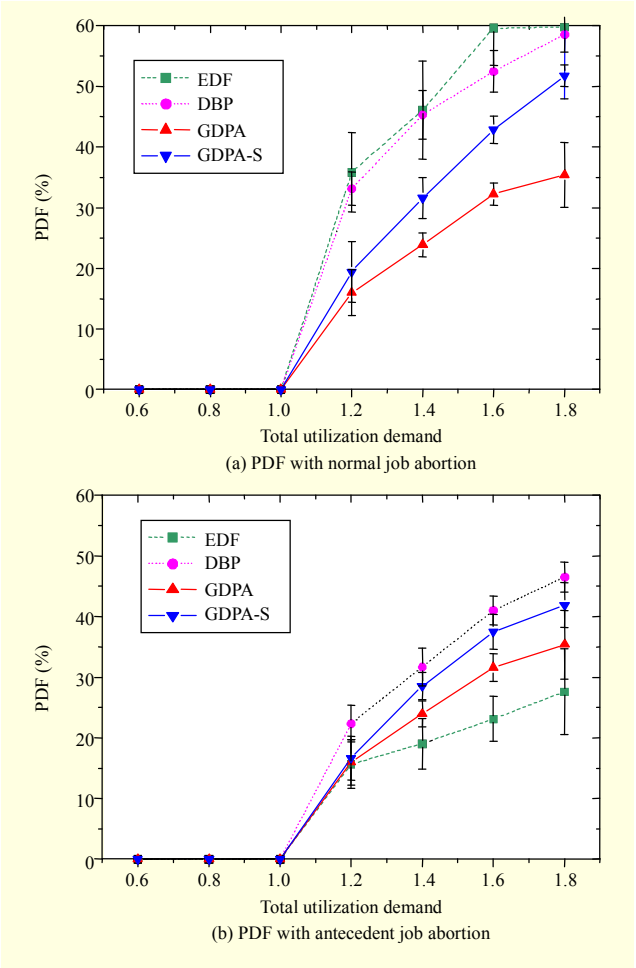


Fig. 2. Performances with tasks constrained (1, 1)-firm deadlines.

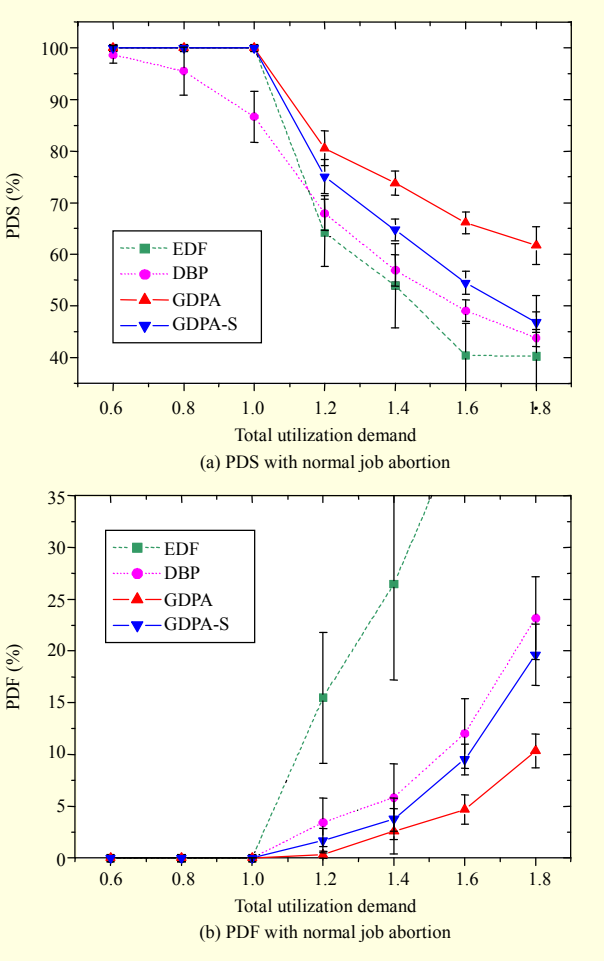


Fig. 3. Performances with tasks constrained (m, k)-firm deadlines.

1. Performance with (1, 1)-Firm Deadlines

We considered tasks having c_i =1 and p_i =10 where the total utilization demand is varied from 0.6 to 1.8. In all the figures shown in this paper, the error bar around each data point represents the 95% confidence interval. Figure 2 shows that all algorithms support a 0% PDF when the total utilization demand is less than or equal to one, that is, when the system is under-loaded.

DBP also behaves in the same manner as EDF when all tasks have the same distance. Since we considered (1, 1)-firm deadlines, all tasks had the same distance, that is, one, in the under-loaded situation. In the over-loaded situation, however, performances differ. GDPA and GDPA-S show relatively a low PDF and DBP shows the highest PDF.

The distances of tasks scheduled by DBP varied over time from 0 to 1 in the over-loaded situation. The PDF of EDF was highly dependent on the selection of a job abortion policy, whereas GDPA and GDPA-S were highly robust to the choice of job abortion policy. The antecedent job abortion policy gave a higher PDF to EDF than the normal one. This implies that a high PDF of EDF can be obtained only when it is allowed to antecedently determine whether the job will meet or miss its deadline. In the case of (1, 1)-firm deadlines, the PDS is simply (100−PDF)%, so all PDS figures are omitted.

Note that GDPA and GDPA-S show a guaranteed PDF (and PDS) in the under-loaded case and simultaneously a high PDF in the over-loaded case with regular hard real-time tasks. In this sense, GDPA and GDPA-S are generic real-time scheduling algorithms that consider both (m, k)-firm deadlines and traditional real-time constraints which can be represented as (1, 1)-firm deadlines.

2. Performance with (m, k)-Firm Deadlines

We considered tasks having c_i and p_i which are randomly generated with uniform distribution in the range [1, 0.8p_i] and [2, 30], respectively, while the total utilization demand is varied from 0.6 to 1.8. Three different (m, k)-firm deadlines, that is, (2, 3), (2, 4), and (1, 2), were randomly assigned to the

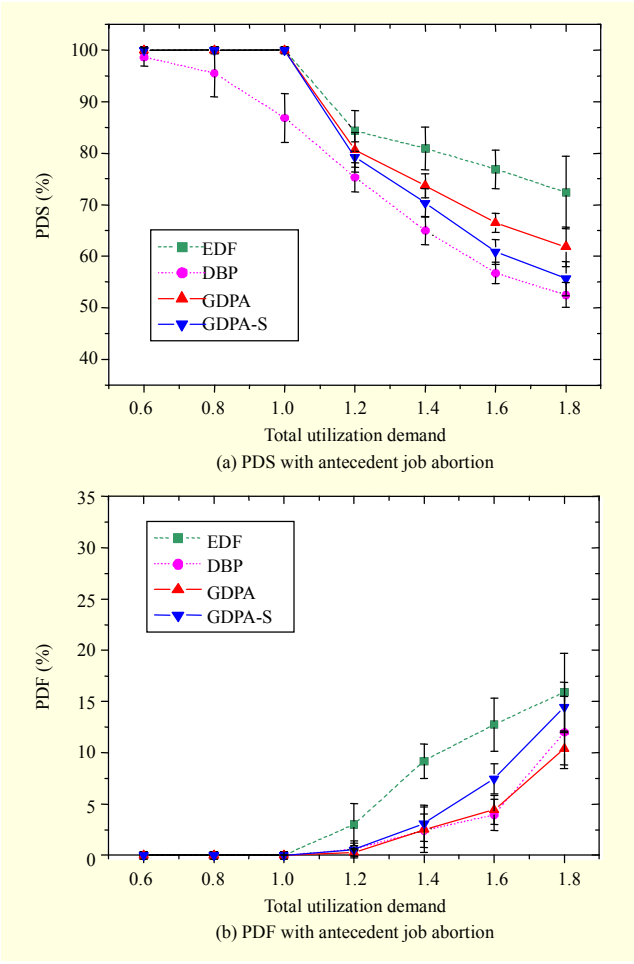


Fig. 4. Performance with tasks constrained (m, k)-firm deadlines.

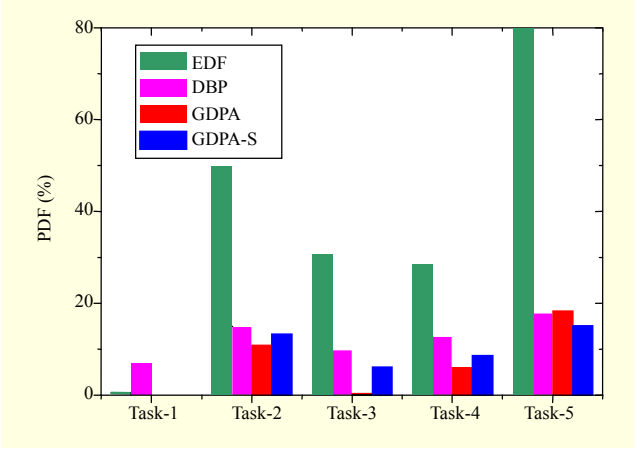


Fig. 5. PDFs of individual tasks.

generated tasks. While the total utilization demand was varied from 0.6 to 1.8, the PDS and PDF were observed.

Figures 3 and 4 show that EDF, GDPA, and GDPA-S support 100% PDS and 0% PDF when the total utilization demand is less than or equal to one, that is, when the system is under-loaded. DBP, however, shows the lowest PDS in the under-loaded situation. Moreover, DBP does not provide any guarantee on its timeliness performance in the under-loaded situation, as opposed to GDPA and GDPA-S. In the over-loaded case, GDPA and GDPA-S show a higher PDS and a lower PDF than DBP. The PDF of EDF is much higher than that of the other algorithms. On the other hand, the PDS of EDF is highly dependent on the choice of job abortion policy, as shown in section VI.1.

We observed that GDPA and GDPA-S provide a guaranteed PDF (and PDS) in the under-loaded situation and, simultaneously, a high PDF in the over-loaded situation with real-time tasks constrained with (m, k)-firm deadlines.

Figure 5 shows the PDF of individual tasks when the total utilization demand is 1.6, that is, an over-loaded situation. The task set contains five tasks characterized with (29, 1, 2, 4), (7, 4, 1, 2), (16, 2, 2, 3), (5, 2, 2, 4), and (23, 10, 1, 2). The PDFs of each task under EDF are observed to be much higher than those of the others, as expected. Interestingly, we observe that the per-task PDFs of DBP are relatively uniform compared to those of GDPA and GDPA-S. This is because DBP makes scheduling decisions without considering differences in execution times, or inter-arrival times, of non-identical tasks. Regardless of each task's temporal characteristics, DBP assigns tasks the same chance of being selected based on their distances. On the other hand, the per-task PDFs of GDPA and GDPA-S are non-uniform. Specifically, T_1 and T_3, which have relatively small utilization demands of 1/29 and 2/16, respectively, have a smaller probability of dynamic failure than T_2 and T_5, which have relatively large utilization demands of 4/7 and 10/23, respectively. This is because GDPA and GDPA-S assign higher chances of execution to tasks which have smaller utilization, that is, shorter execution times and/or longer relative deadlines, resulting in rare occurrences of deadline-misses and dynamic failures. This implies that GDPA and GDPA-S utilize each task's temporal characteristics as well as its distance for scheduling decisions.

VII. Conclusions

We presented two guaranteed dynamic priority assignment schemes, GDPA and GDPA-S, for multiple tasks with (m, k)-firm deadlines. GDPA and GDPA-S have a two-fold scheduling objective: providing a bounded probability of dynamic failures and maximizing the probability of deadline satisfactions. We analytically confirmed that the two proposed algorithms provide a guarantee of zero probability of dynamic failures when the system is under-loaded. In addition, we showed that GDPA and GDPA-S improve the quality of service by maximizing the probability of deadline satisfactions. Our experimental studies validated our analytical results and

showed the effectiveness of the proposed schemes.

Examples for further research include relaxing the task arrival model. Implementations of GDPA and GDPA-S for multimedia stream services are also definite future directions.

References

[1] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines," *IEEE Trans. Comput.*, vol. 44, no. 2, 1995, pp. 1443-1451.

[2] G. Bernat and R. Cayssials, "Guaranteed On-Line Weakly-Hard Real-Time Systems," *IEEE Real-Time System Symp.*, 2001.

[3] P. Ramanathan, "Overload Management in Real-Rime Control Applications Using (m, k)-Firm Guarantee," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 6, June 1999, pp. 549-559.

[4] C.L. Liu, and J.W Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, 1973, pp. 46-61.

[5] M. Hamdaoui and P. Ramanathan, "Evaluating Dynamic Failure Probability for Streams with (m, k)-Firm Deadlines," *IEEE Trans. Comput.*, vol. 46, no. 12, Dec. 1997, pp. 1325-1337.

[6] G. Bernat, A. Burns, and A. Llamosi, "Weakly-Hard Real-Time Systems," *IEEE Trans. Comput.*, vol. 50, no. 4, Apr. 2001, pp. 308-321.

[7] J. Chen et al., "Scalability and QoS Guarantee for Streams with (m,k)-Firm Deadline," *Computer Standards and Interfaces*, vol. 28, issue 5, Jun. 2006, pp. 560-571.

[8] Z. Wang, J.M. Chen, and Y.X. Sun, "An Integrated DBP for Streams with (m,k)-Firm Real-Time Guarantee," *J. Zhejiang University Science*, vol. 5, no. 7, 2004, pp. 816-826,

[9] E. Poggi et al., "Matrix-DBP For (m, k)-Firm Real-Time Guarantee," *Proc. Real-Time and Embedded System*, 2003, pp. 457-482.

[10] J. Chen et al,, "Equivalent Matrix DBP for Streams with (m,k)-Firm Deadline," *IEEE Int. Symp. Ind. Electron.*, vol. 1, May 2004, pp. 675-680.

[11] G. Quan and X. Hu, "Enhanced Fixed-Priority Scheduling with (m,k)-Firm Guarantee," *IEEE RTSS*, 2000, pp. 79-88.

[12] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, 1997.

[13] J. Lin and A.M.K. Cheng, "Maximizing Guaranteed QoS in (m,k)-Firm Real-Time Systems," *IEEE Int. Conf. Embedded and Real-Time Computing*, 2006.

[14] J. Li, Y. Song, and F. Simonot-Lion, "Providing Real-Time Applications with Graceful Degradation of QoS and Fault Tolerance According to (m,k)-Firm Model," *IEEE Trans. Ind. Inf.*, May 2006, pp. 112-119.

[15] F. Flavia et al., "Optimal On-Line (m,k)-Firm Constraint Assignment for Real-Time Control Tasks Based on Plant State Information," *IEEE Int. Conf. Emerging Technologies and Factory Automation*, 2008, pp. 908-915.

[16] A. Striegel and G. Manimaran. "Best-Effort Scheduling of (m, k)-Firm Real-Time Streams in Multihop Networks," *Proc. 15 IPDPS 2000 Workshops Parallel Distrib. Process.*, 2000, pp 743-751.

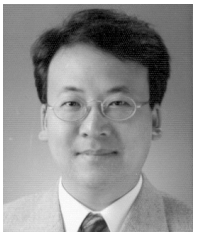
[17] S. Lai, B. Ravindran, and H. Cho, "On Scheduling Soft Real-Time Tasks with Lock-Free Synchronization for Embedded Devices," *ACM Symp. Applied Computing (SAC)*, Mar. 2009, pp. 1685-1686.

[18] T. Wu and S. Jin, "Weakly Hard Real-Time Scheduling Algorithm for Multimedia Embedded System on Multiprocessor Platform," *IEEE Int. Conf. Ubi-Media Computing*, 2008.

[19] L. Niu and G. Quan, "A Hybrid Static/Dynamic DVS Scheduling for Real-Time Systems with (m, k)-Guarantee," *IEEE Int. Real-Time Systems Symp.*, 2006.

[20] C.D. Locke, *Best Effort Decision Making for Real-Time Scheduling*, PhD Thesis, Carnegie Mellon University, 1986.

Hyeonjoong Cho is an assistant professor in the Department of Computer and Information Science at Korea University. His research focuses on real-time systems, embedded systems, industrial field buses, wireless sensor networks, etc. Before he joined Korea University in 2009, he worked as a senior researcher in ETRI. He received the PhD degree in computer engineering from Virginia Polytechnic Institute and State University in 2006. He received his BS degree from Kyungpook National University (1996) and MS degree from POSTECH (1998), Korea. He had worked for Samsung Electronics as a senior software engineer before pursuing his PhD degree.



Yongwha Chung received the BS and MS degrees from Hanyang University, Korea, in 1984 and 1986, respectively. He received the PhD degree from the University of Southern California, USA, in 1997. He worked for ETRI from 1986 to 2003 as a team leader. Currently, he is a professor in the Department of Computer and Information, Korea University. His research interests include multimedia, security, and performance optimization.

Daihee Park received his BS degree in mathematics from Korea University, Korea, in 1982, and his PhD degree in computer science from the Florida State University, USA, in 1992. He joined Korea University in 1993, where he is currently a professor in the Department of Computer and Information Science. His research interests include data mining and intelligent database.