12-1-2009

Multiprocessor Real-Time Scheduling Considering Concurrency and Urgency

Jinkyu Lee
KAIST, South Korea, jinkyu@cps.kaist.ac.kr

Arvind Easwaran
Polytechnic Institute of Porto (ISEP-IPP), Portugal, aen@isep.ipp.pt

Insik Shin
KAIST, South Korea, insik.shin@cs.kaist.ac.kr

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Special Issue on the Work-in-Progress (WIP) Session at the 2009 IEEE Real-Time Systems Symposium (RTSS), Washington, D.C., December 1-4, 2009. http://sigbed.seas.upenn.edu/vol7_num1.html

Multiprocessor Real-Time Scheduling Considering Concurrency and Urgency

Abstract
It has been widely studied how to schedule real-time tasks on multiprocessor platforms. Several studies find optimal scheduling policies for implicit deadline task systems, but it is hard to understand how each policy utilizes the two important aspects of scheduling real-time tasks on multiprocessors: inter-job concurrency and job urgency. In this paper, we introduce a new scheduling policy that considers these two properties. We prove that the policy is optimal for the special case when the execution time of all tasks are equally one and deadlines are implicit, and observe that the policy is a new concept in that it is not an instance of Pfair or ERfair. It remains open to find a scheduliability condition for general task systems under our scheduling policy.

Disciplines
Computer Engineering | Engineering

Comments
Special Issue on the Work-in-Progress (WIP) Session at the 2009 IEEE Real-Time Systems Symposium (RTSS), Washington, D.C., December 1-4, 2009. http://sigbed.seas.upenn.edu/vol7_num1.html

Multiprocessor Real-Time Scheduling Considering Concurrency and Urgency

Jinkyu Lee*, Arvind Easwaran†, Insik Shin* and Insup Lee‡

*Dept. of Computer Science, KAIST, South Korea
†IPP-HURRAY! Research Group, Polytechnic Institute of Porto (ISEP-IPP), Portugal
‡Dept. of Computer and Information Science, University of Pennsylvania, USA
jinkyu@cps.kaist.ac.kr; aen@isep.ipp.pt; insik.shin@cs.kaist.ac.kr; lee@cis.upenn.edu

***Abstract*—It has been widely studied how to schedule real-time tasks on multiprocessor platforms. Several studies find optimal scheduling policies for implicit deadline task systems, but it is hard to understand how each policy utilizes the two important aspects of scheduling real-time tasks on multiprocessors: inter-job concurrency and job urgency. In this paper, we introduce a new scheduling policy that considers these two properties. We prove that the policy is optimal for the special case when the execution time of all tasks are equally one and deadlines are implicit, and observe that the policy is a new concept in that it is not an instance of Pfair or ERfair. It remains open to find a scheduliability condition for general task systems under our scheduling policy.**

I. Introduction

Real-time schedulability analysis have been studied for achieving predictability on satisfying timing constraints. In particular, scheduling policies for uniprocessor have been extensively studied. EDF [1] and DM [2] are optimal dynamic- and static-priority scheduling policies for preemptive scheduling of periodic and sporadic tasks, respectively. While uniprocessor scheduling has matured over years, finding optimal scheduling policies for general task systems on multiprocessor platforms is still an open problem. Some studies (e.g., [3], [4], [5]) focused on adapting existing uniprocessor scheduling to multiprocessor scheduling, but they do not optimally utilize the processing capacity. This is because uniprocessor policies are not designed to efficiently handle concurrent executions. Several other studies (e.g., [6], [7], [8], [9], [10], [11]) have been introduced that generate optimal schedules for implicit deadline task systems on multiprocessor platforms, but some of these algorithms suffer from high preemption overhead. Further, none of them is able to preserve optimality for more general task systems (such as constrained deadline task systems). We believe this limitation in the state-of-art arises from the fact that it is difficult to understand how existing policies treat some of the important aspects of scheduling real-time tasks on multiprocessor platforms. Therefore, in this paper, we design a novel scheduling policy that clearly differs from existing policies in this regard. That is, it explicitly uses important aspects of scheduling like "job urgency" and "inter-job concurrency" to prioritize jobs.

Handling the trade-off between "job urgency" and "inter-job concurrency" is, in our opinion, the key to efficient scheduling on multiprocessor platforms. This can be explained as follows. To maximize the number of concurrently executing jobs, it is desirable to delay the finishing time of jobs so that more unfinished jobs are available for scheduling. For instance, a policy which gives "higher priority to jobs with longer remaining execution time" implements this concept. However, in order to meet hard real-time requirements it is also important to finish jobs by their deadlines. For instance, a policy which gives "higher priority to jobs with earlier deadline" implements this concept. Therefore, one approach for allowing a trade-off between these concepts would be to simultaneously consider the remaining execution time and deadline of jobs.

In this work, we first consider a simple and intuitive scheduling policy based on the above discussion (called *Dynamic Density First (DDF)*). The dynamic density of a job is defined as its remaining execution time divided by the time to deadline. Note that this parameter changes continuously over a job's lifetime. DDF assigns a higher priority to a job with a larger dynamic density. However, we observed that such a simple (and in some sense crude) strategy does not offer a very fine-grained trade-off between urgency and concurrency. For example, consider two jobs; one with a higher dynamic density and a longer time to deadline, and the other with a lower dynamic density but a shorter time to deadline. As DDF will schedule the former job, it could lead to a situation where the latter job misses its deadline eventually. Therefore, it entails another more refined scheduling strategy.

In this paper, we introduce a new scheduling policy extending DDF. We observe and prove that DDF is an optimal multiprocessor scheduler for implicit deadline tasks, when their execution times are all equally one. DDF is not optimal for general tasks with arbitrary execution times however. Looking at how DDF fails to schedule such general tasks, we observe that some jobs are executed earlier than they should be. Its implication is that an optimal schedule can be obtained from a DDF schedule if we can delay the

execution of "some" jobs. Reflecting this, we introduce a new scheduling policy called LADD (Lagging And Dynamic Density). A job is said to be *lagging* if it has a longer remaining execution time when compared to some nominal value (we describe this nominal value later in the paper). In LADD, jobs are classified into two groups: a group of lagging jobs and another group of non-lagging jobs. All jobs in the lagging group have a higher priority than those in the non-lagging group. Further, jobs in the same group are scheduled using DDF policy. LADD favors lagging jobs first and then jobs with higher dynamic density; it essentially delays the execution of non-lagging jobs. Our goal is to investigate the performance of LADD. In this paper, we show that for the special case where the execution times of all tasks are one, LADD produces the same (optimal) schedule as the one by DDF. We are currently working on finding a schedulability condition for general tasks systems under LADD.

The contributions of this paper are as follows: we introduce a new scheduling policy that considers both concurrency and urgency; we prove the optimality of the policy for a special task system; and we observe that the policy is not an instance of Pfair [6] or ERfair [8].

Task Model. We assume a constrained deadline sporadic task model [12]. In this model, a task τ_i is specified as (T_i, C_i, D_i), where T_i is the minimum separation, C_i is the worst-case execution time requirement, and D_i is the relative deadline. We assume $C_i \leq D_i \leq T_i$. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. We assume that a single job of a task cannot be executed in parallel. There are m processors in the system, and we assume that $\sum_{\forall j} \frac{C_j}{D_j} \triangleq D_{sys}$ (named as the system static density) is not more than m.

We use $D_i(t)$ and $C_i(t)$ to denote the remaining time to deadline and the remaining execution time, respectively, of a job of τ_i at time t. The dynamic density of a job of τ_i at t is then specified as $\frac{C_i(t)}{D_i(t)}$, and the system dynamic density is $\sum_j \frac{C_j(t)}{D_j(t)} \triangleq D_{sys}(t)$. We express that a job of τ_i is *active* at t when $C_i(t)$ is non-zero. We denote the number of tasks as n, and the number of active jobs at t as $n(t)$.

We consider quantum-based (discrete) systems, and thus the schedule is also quantum-based.

II. Optimality of DDF for $C_i = 1$

In this section, we prove that we can schedule any task set under the following assumptions: (A1) the task set is scheduled by DDF; (A2) the system static density is not more than m; and (A3) execution times of all tasks are one ($C_i = 1$), and thus remaining execution times of all active jobs are also one ($C_i(t) = 1$, $\forall$ active τ_i at t). First, we prove that the system dynamic density does not increase in case of no new arrival of jobs. Second, we prove that the system dynamic density at any time t cannot exceed m in spite of arrival of new jobs.

The first lemma shows that if remaining time to deadline of all jobs are identical, then the system dynamic density cannot increase in case of no new arrival of jobs.

Lemma 1: Assume that the following conditions: (A4) remaining time to deadline of all active jobs at t_0 are identical (*i.e.*, $D_i(t_0) = D(t_0)$); (A5) the system dynamic density at t_0 is not more than m; and (A6) there is no new arrival of jobs in the interval $[t_0, t_1)$. Then, $D_{sys}(t) \leq D_{sys}(t_0)$ for all $t \in [t_0, t_1)$.

Proof: We use mathematical induction.

(The basis) At t_0, $D_{sys}(t_0) \leq D_{sys}(t_0)$.

(The inductive step) We wish to prove the following: if $D_{sys}(t) \leq m$ is true, then $D_{sys}(t+1) \leq D_{sys}(t)$. From (A3) and (A4), we derive $D_{sys}(t) = \frac{n(t)}{D(t)}$ implying $n(t) = D_{sys}(t) \cdot D(t)$. Assuming m jobs are serviced in $[t, t+1)$, we calculate $n(t+1) = n(t) - m$. We then have the results below.

$$D_{sys}(t+1) = \frac{n(t+1)}{D(t+1)} = \frac{D_{sys}(t) \cdot D(t) - m}{D(t) - 1} \leq D_{sys}(t).$$

Here we assumed that m jobs are scheduled in the time interval $[t, t+1)$. If there are less than m active jobs at t, then there is no active job at $t+1$. This means $D_{sys}(t+1) = 0 \leq D_{sys}(t)$, and hence this lemma is always true. ■

We now wish to show that the above lemma holds even when remaining time to deadline of jobs are different. For this purpose, we first show a system dynamic density bounding transformation from a set of jobs with different deadlines to a set of jobs with identical deadline (Lemma 2). Then, in Lemma 3 we prove that Lemma 1 holds even when job deadlines are different.

Lemma 2: Assume $k \cdot \frac{1}{D} = \sum_j \frac{1}{D_j}$ and $D \leq D_j$ for all j. Then, we can derive that $k \cdot \frac{1}{D-t} \geq \sum_{\forall j} \frac{1}{D_j - t}$ for all $0 < t < D$.

Proof:

$$k \cdot \frac{1}{D-t} = k \cdot \frac{1}{D} + k \cdot \frac{1}{D} \cdot \frac{t}{D-t} = \sum_j \frac{1}{D_j} + \sum_j \frac{1}{D_j} \cdot \frac{t}{D-t}$$
$$\geq \sum_j \frac{1}{D_j} + \sum_j \frac{1}{D_j} \cdot \frac{t}{D_j - t} = \sum_j \frac{1}{D_j - t}$$

■

Lemma 3: Assume (A5) shown in Lemma 1, and suppose there is no arrival of new jobs in $[t, t+1)$. Then, we can derive that $D_{sys}(t+1) \leq D_{sys}(t)$.

Proof: Without loss of generality, we sort the index of jobs by remaining time to deadline at t as follows.

$$D_1(t) \leq ... \leq D_m(t) \leq D_{m+1}(t) \leq ... \leq D_{n(t)}(t) \quad (1)$$

We construct a set of new jobs satisfying the following: (a) the system dynamic density at t of the new jobs is same as that of the original jobs; (b) the m most urgent jobs (as per DDF) from the new set is the same as that in the original set; and (c) other new jobs except the m most urgent jobs are the same as the m^{th} urgent job in the original set. Note that, with this transformation, the number of jobs in the new set at time t is no more than that in the original set. Thus, the set of new jobs can be expressed as follows.

$$\begin{aligned} D'_1(t) \leq ... \leq D'_m(t) = D'_{m+1}(t) = ... = D'_{n'(t)}(t), \\ \text{where } D'_1(t) = D_1(t), ..., D'_m(t) = D_m(t) \\ \text{and } \sum_{j=1}^{n'(t)} \frac{1}{D'_j(t)} = \sum_{j=1}^{n(t)} \frac{1}{D_j(t)}, n'(t) \leq n(t) \end{aligned} \quad (2)$$

During $[t, t+1)$ the m most urgent jobs are serviced, so the system dynamic density at $t+1$ of the set described in Eq. (2) is $\sum_{j=m+1}^{n'(t)} \frac{1}{D'_j(t)-1} = \frac{n'(t)-m}{D'_{m+1}(t)-1}$. Here we know that $D'_{m+1}(t)$ is equal to or less than any of $D_{m+1}(t), ..., D_{n(t)}(t)$. By Lemma 2, the system dynamic density at $t+1$ of the set described in Eq. (2) is not less than that of the set described in Eq. (1).

We now define another set of new identical jobs as follows:

$$\begin{aligned} D^*_1(t) = ... = D^*_m(t) = D^*_{m+1}(t) = ... = D^*_{n^*(t)}(t), \\ \text{where } \sum_{j=1}^{n^*(t)} \frac{1}{D^*_j(t)} = \sum_{j=1}^{n(t)} \frac{1}{D_j(t)}, n^*(t) = n'(t) \end{aligned} \quad (3)$$

Note these jobs also have the system dynamic density at t same as that of the original job set. Further, it is easy to see that $D^*_{m+1}(t) \leq D'_{m+1}(t)$, and thus the system dynamic density at $t+1$ of the set described in Eq. (3) $\left(\sum_{j=m+1}^{n^*(t)} \frac{1}{D^*_j(t)-1}\right)$ is equal to or larger than that of the set described in Eq. (2) $\left(\sum_{j=m+1}^{n'(t)} \frac{1}{D'_j(t)-1}\right)$. Therefore, by Lemma 1 we get,

$$\begin{aligned} \sum_{j=1}^{n(t+1)} \frac{1}{D_j(t+1)} \leq \sum_{j=1}^{n'(t+1)} \frac{1}{D'_j(t+1)} \\ \leq \sum_{j=1}^{n^*(t+1)} \frac{1}{D^*_j(t+1)} \leq \sum_{j=1}^{n^*(t)} \frac{1}{D^*_j(t)} = \sum_{j=1}^{n(t)} \frac{1}{D_j(t)}, \end{aligned}$$

and thus we conclude that $D_{sys}(t+1)$ is not more than $D_{sys}(t)$[1]. Similar to Lemma 1, it does not affect the proof that there can be less than m active jobs at t. ■

[1]In this lemma, the defined new jobs described in Eq. (2) and (3) may have non-integer deadlines, which means a continuous system is allowed. However, since the worst case of $D_{sys}(t+1)$ for both discrete and continuous systems is not less than that for discrete systems, this lemma holds for discrete systems.

It now remains to prove that Lemma 1 holds even when new jobs are released in the interval of interest. For this purpose, we first prove that when the remaining time to deadline of a job of τ_i becomes zero, the system dynamic density is at most $m - \left(\frac{1}{D_i}\right)$. The proof technique of the following lemma is similar to that of Lemma 3.

Lemma 4: Assume (A5) in Lemma 1, and suppose there is no arrival of new jobs in $[t, t+D_1(t)]$, where $D_1(t)$ is the remaining time to deadline of the most urgent job at t. We denote the number of most urgent jobs at t as N (note all have deadline at $t+D_1(t)$). Then we conclude $D_{sys}(t+D_1(t)) \leq D_{sys}(t) - \frac{N}{D_1(t)}$.

Proof: Without loss of generality, we sort the index of jobs by the remaining time to deadline at t.

$$\begin{aligned} D_1(t) = ... = D_N(t) \leq D_{N+1}(t) \leq ... \leq D_{m \cdot D_1(t)}(t) \\ \leq D_{m \cdot D_1(t)+1}(t) \leq ... \leq D_{n(t)}(t) \end{aligned} \quad (4)$$

We construct a set of new jobs in a similar way to Lemma 3, as follows:

$$\begin{aligned} D'_1(t) = ... = D'_N(t) \leq D'_{N+1}(t) \leq ... \leq D'_{m \cdot D'_1(t)}(t) \\ = D'_{m \cdot D'_1(t)+1}(t) = ... = D'_{n'(t)}(t), \\ \text{where } D'_1(t) = D_1(t), ..., D'_{m \cdot D'_1(t)}(t) = D_{m \cdot D_1(t)}(t) \\ \text{and } \sum_{j=1}^{n'(t)} \frac{1}{D'_j(t)} = \sum_{j=1}^{n(t)} \frac{1}{D_j(t)}, n'(t) \leq n(t) \end{aligned} \quad (5)$$

By Lemma 2, the system dynamic density at $t' \in [t, t+D_1(t)]$ of the set described in Eq. (5) is equal to or larger than that of the set described in Eq. (4).

We now define another set of new identical jobs similar to Lemma 3, but in this case we do not change the N most urgent jobs $\{D_1(t), ..., D_N(t)\}$.

$$\begin{aligned} D^*_1(t) = ... = D^*_N(t) \leq D^*_{N+1}(t) = ... = D^*_{m \cdot D^*_1(t)}(t) \\ = D^*_{m \cdot D'_1(t)+1}(t) = ... = D^*_{n^*(t)}(t), \\ \text{where } D^*_1(t) = D_1(t), ..., D^*_N(t) = D_N(t), \\ \text{and } \sum_{j=1}^{n^*(t)} \frac{1}{D^*_j(t)} = \sum_{j=1}^{n(t)} \frac{1}{D_j(t)}, n^*(t) = n'(t) \end{aligned} \quad (6)$$

We can calculate the system dynamic density at t of the set described in Eq. (6) by $U^*_{sys}(t) = \frac{N}{D^*_1(t)} + \frac{n^*(t)-N}{D^*_{N+1}(t)}$. During $[t, t+D^*_1(t)]$, $D^*_1(t) \cdot m$ jobs are serviced, and thus, using $U^*_{sys}(t)$, we calculate the system dynamic density at $t+D_1(t)$ of this set as follows.

$$U^*_{sys}(t + D_1(t)) = \frac{n^*(t) - D^*_1(t) \cdot m}{D^*_{N+1}(t) - D^*_1(t)}$$
$$= \frac{n^*(t) - D^*_1(t) \cdot U^*_{sys}(t)}{n^*(t) - D^*_1(t) \cdot m} \left(U^*_{sys}(t) - \frac{N}{D_1(t)} \right)$$
$$\leq U^*_{sys}(t) - \frac{N}{D_1(t)}$$

Using arguments identical to Lemma 3 we can conclude that $D_{sys}(t + D_1(t))$ is equal to or smaller than $D_{sys}(t) - \frac{N}{D_1(t)}$.

■

Using the previous lemmas, we finally have the following theorem.

Theorem 1: DDF can schedule any task set which satisfies (A2) and (A3).

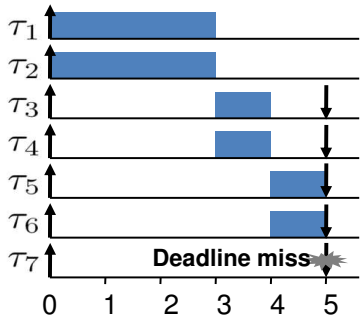
Proof: Assume that the system dynamic density at t is equal to or less than m. By Lemma 4, $D_{sys}(t+D_1(t))$ is not more than $D_{sys}(t) - \frac{N}{D_1(t)}$. At $t + D_1(t)$, we have enough slack in the system dynamic density to accommodate the arrival of a new job of task τ_1. Since we assume constrained deadline tasks, we then guarantee that the arrival of new jobs of τ_1 cannot make the system dynamic density larger than m. Since we know the system dynamic density cannot increase without arrival of new jobs from Lemma 3, it is enough to look at points when remaining time to deadline of any job becomes zero. Since the system dynamic density at the start is not larger than m (system static density is at most m), we then guarantee that the system dynamic density never exceeds m.

At any time, there are at most m urgent jobs (*i.e.*, $C_i(t) = D_i(t)$), and these jobs have the highest priorities. Therefore, DDF can schedule any task set which satisfies (A2) and (A3). ■

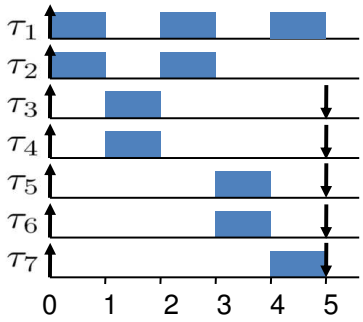
A direct corollary of the above theorem is that DDF is optimal for implicit deadline task systems when execution time of all tasks are equally one. As an aside, note that for this specific task system, the schedule generated by DDF is identical to that generated by global EDF.

III. Toward Optimality of LADD

We have proved in Section II that DDF is an optimal scheduling policy when the execution times of all tasks are equally one and deadlines are equal to periods. But it can be easily verified that DDF is not optimal without such an assumption on the execution time. Consider a task system that comprises of seven tasks as follows: $\tau_1 = \tau_2 = (14, 7, 14)$, $\tau_3 = \tau_4 = \tau_5 = \tau_6 = \tau_7 = (5, 1, 5)$. This task set is scheduled on a multiprocessor platform that consists of two processors. As shown in Figure 1(a), if we apply DDF, τ_7 cannot be scheduled until its deadline $t = 5$. This



(a) The original task set



(b) The alternative task set

Figure 1. Schedule under DDF

scheduling failure comes from early execution of τ_1 and τ_2. If we substitute the original set by a new task set where $\tau'_1 = \tau'_2 = (2, 1, 2)$ and other tasks are the same, DDF produces a feasible schedule as shown in Figure 1(b). Since any feasible schedule of the new task set can be used for the original task set, we can see that DDF's schedule becomes a feasible schedule by postponing the execution of τ_1 and τ_2.

To improve DDF, we must answer the question "when to delay the execution of jobs and which ones." For this we consider a parameter called the expected remaining execution time of task τ_i (denoted as $C^E_i(t)$). If a job of τ_i is ideally scheduled with a rate of $\frac{C_i}{D_i}$, its remaining execution time at t becomes $C^E_i(t)$, which means $C^E_i(t) = \frac{C_i}{D_i} \cdot D_i(t)$. A job is said to be *lagging* if $C_i(t)$ is strictly larger than $C^E_i(t+1)$. The intuitive meaning of a *lagging* job is that if the job is not serviced in $[t, t+1)$, its remaining execution time becomes larger than its expected remaining execution time at $t + 1$. Using this concept of lagging, we now introduce a new scheduling policy called LADD (Lagging And Dynamic Density). In LADD, we divide jobs into two groups: lagging jobs and non-lagging jobs. At every t, we schedule m lagging jobs which have higher dynamic density at t (scheduled by DDF). If there are less than m lagging jobs, we schedule non-lagging jobs also prioritized using DDF.

In the following theorem, we prove that a schedule of LADD is the same as that of DDF for task sets where execution times of all tasks are identically one[2].

Theorem 2: LADD can schedule any task set, which satisfies (A2) and (A3) shown in Section II.

Proof: Any active jobs at t satisfy the following.

$$C_i^E(t+1) = \frac{C_i}{D_i} \cdot (D_i(t+1)) = \frac{1}{D_i} \cdot (D_i(t) - 1) < 1 = C_i(t),$$

which means any active jobs are lagging. So, LADD produces the same schedule as DDF. By Theorem 1, we conclude that LADD can schedule any task set, which satisfies (A2) and (A3). In other words, LADD is also optimal for any implicit deadline task system where execution time of all tasks are equally one. ■

In the following observation, we claim that LADD is a new scheduling concept.

Observation 1: LADD is not an instance of Pfair or ERfair.

Proof: We provide a example. Consider a task system comprised of six tasks as follows: $\tau_1 = (157, 66, 157)$, $\tau_2 = (667, 174, 667)$, $\tau_3 = (867, 162, 867)$, $\tau_4 = (132, 127, 132)$, $\tau_5 = (878, 120, 878)$, $\tau_6 = (31, 1, 31)$. In this system, $m = 2$ and $D_{sys} < 2$. When we apply LADD, *lag* (as defined in [6]) of τ_5 at $t = 8$ is strictly larger than 1.0. ■

IV. Conclusion

We present a new multiprocessor scheduling policy that offers a fine-grained trade-off between concurrency and urgency. We prove that the proposed scheduling policy is optimal for implicit deadline task systems where execution time of all tasks are equally one. We also observe that our scheduling policy is not an instance of Pfair or ERfair.

Our future work involves deriving a schedulability condition for general task systems under LADD. Another direction of our future work is to find the theoretical bound on the number of preemptions and migrations. We also plan to compare overhead of LADD with that of other scheduling algorithms (*e.g.*, EKG [9] and LLREF [7]) through simulation and/or analysis.

Acknowledgement

This research was supported in part by IT R&D program of MKE/KEIT of Korea [2009-KI002090, Development of Technology Base for Trustworthy Computing], National Research Foundation of Korea (2009-0086964), and KAIST ICC, KIDCS, KMCC, and OLEV grants.

This work was also partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the European Commission through grant ArtistDesign ICT-NoE-214373.

[2]We conjecture that the scheduling policy of non-lagging jobs does not affect optimality.

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

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

[3] S. Cho, S.-K. Lee, S. Ahn, and K.-J. Lin, "Efficient real-time scheduling algorithms for multiprocessor systems," *IEICE Trans. on Communications*, vol. E85–B, no. 12, pp. 2859–2867, 2002.

[4] A. Srinivasan and S. Baruah, "Deadline-based scheduling of periodic task systems on multiprocessors," *Information Processing Letters*, vol. 84, no. 2, pp. 93–98, 2002.

[5] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *RTSS*, 2001.

[6] S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[7] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS*, 2006.

[8] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *ECRTS*, 2000, pp. 35–43.

[9] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *RTCSA*, 2006, pp. 322–334.

[10] K. Funaoka, S. Kato, and N. Yamasaki, "Work-conserving optimal real-time scheduling on multiprocessors," in *ECRTS*, 2008.

[11] B. Andersson and K. Bletsas, "Sporadic multiprocessor scheduling with few preemptions," in *ECRTS*, 2008, pp. 243–252.

[12] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS*, 1990.