ELSEVIER

Information Processing Letters 84 (2002) 93–98

Information Processing Letters

www.elsevier.com/locate/ipl

Deadline-based scheduling of periodic task systems on multiprocessors ☆

Anand Srinivasan, Sanjoy Baruah *

The University of North Carolina, Department of Computer Science, CB 3175, Chapel Hill, NC 27599-3175, USA

Received 22 May 2001; received in revised form 10 January 2002

Communicated by F.Y.L. Chin

Abstract

We consider the problem of scheduling periodic task systems on multiprocessors and present a deadline-based scheduling algorithm for solving this problem. We show that our algorithm successfully schedules on m processors any periodic task system with utilization at most $m^2/(2m-1)$.

Keywords: Real-time systems; Multiprocessor scheduling; Periodic task systems; Feasibility analysis; Earliest deadline first

1. Introduction

Over the years, the preemptive periodic task model [11,12] has proven remarkably useful for the modeling of recurring processes that occur in hard-real-time computer application systems. Accordingly, much effort has been devoted to the development of a comprehensive theory dealing with the scheduling of systems comprised of such independent periodic real-time tasks. Particularly in the uniprocessor context—in environments in which all hard-real-time jobs generated by all the periodic tasks that comprise the hard-real-time application system must execute on a single shared processor—there now exists a wide body of results (necessary and sufficient feasibility tests, optimal scheduling algorithms, efficient implementations of these algorithms, etc.) that facilitate the application systems designer who is able to model his or her real-time application system as a collection of independent preemptive periodic real-time tasks. Some of these results have been extended to the multiprocessor context—environments in which there are several identical processors available upon which the real-time jobs may be executed.

The periodic task model. In the periodic model of hard real-time tasks, a *task* $\tau_i = (C_i, T_i)$ is characterized by two parameters—an execution requirement C_i and a period T_i—with the interpretation that the task generates a *job* at each integer multiple of T_i, and each such job has an execution requirement of C_i units, and must complete by a deadline equal to the next integer multiple of T_i. A periodic task system consists of several such periodic tasks that are to execute on a specified processor architecture. The utilization of periodic

☆ Supported in part by the National Science Foundation (Grant Nos. CCR-9988327, and ITR-0082866).

* Corresponding author.
E-mail address: baruah@cs.unc.edu (S. Baruah).

0020-0190/02/$ – see front matter
PII: S0020-0190(02)00231-4

task τ_i is defined as the ratio of its execution requirement to its period. The utilization of a periodic task system is the sum of the utilizations of all the periodic tasks contained in it.

We assume that each job is independent in the sense that it does not interact in any manner (accessing shared data, exchanging messages, etc.) with other jobs of the same or another task. We also assume that the model allows for job *preemption*; i.e., a job executing on a processor may be preempted prior to completing execution, and its execution may be resumed later, at no cost or penalty.

Identical multiprocessor platform. In this paper, we study the scheduling of periodic task systems on m ($\geqslant 1$) identical multiprocessors. We assume that interprocessor migration of jobs is permitted—i.e., a job that is executing upon a processor may be preempted and may later resume execution on a different processor. We do *not* permit job-level parallelism, i.e., a job executes on at most one processor at any instant of time.

Deadline-based scheduling. *Run-time scheduling* is the process of determining, during the execution of a real-time application system, which job[s] should be executed at each instant in time. Run-time scheduling algorithms are typically implemented as follows: at each time instant, assign a *priority* to each active[1] job, and allocate the available processors to the highest-priority jobs.

One of the most popular algorithms used for priority assignment in run-time scheduling is the *earliest deadline first* scheduling algorithm (EDF) [12,5]. In EDF, jobs are assigned priorities in inverse proportion to their deadlines—the earlier the deadline, the higher the priority. EDF is known to be an *optimal* scheduling algorithm for uniprocessors; unfortunately, EDF is *not* optimal on multiprocessors. There are nevertheless significant advantages to using EDF for scheduling on multiprocessors if possible; while it is beyond the scope of this document to describe in detail all these advantages, some important ones are listed below.

- Very efficient implementations of EDF have been designed (see, e.g., [13]).
- It can be shown that when a set of jobs is scheduled using EDF, then the total number of *preemptions* is bounded from above by the number of jobs in the set (and consequently, the total number of *context switches* is bounded at twice the number of jobs).
- In a similar way, it can be shown that the total number of *interprocessor job migrations* is bounded from above by the number of jobs.

The last two properties apply to a broader class of algorithms, discussed later in this paper.

There have been two approaches towards scheduling of periodic tasks on multiprocessors: *partitioning* and *global scheduling*. In the partitioning approach, the tasks are statically partitioned among the processors, i.e., each task is assigned to a processor and is always executed on it. Note that this approach is not work-conserving in the sense that some processors may be left idle even if there are unfinished jobs (assigned to other processors). On the other hand, in global scheduling, the tasks are not partitioned and are put in a single global queue. The scheduler selects the m highest priority jobs for execution, where m is the number of processors. Unfortunately, well-known uniprocessor scheduling algorithms such as EDF perform poorly in this scenario. In fact, in [6], it was shown that EDF can produce arbitrarily low processor utilization. Pfair and ERfair scheduling algorithms [3,4,1] are optimal in this scenario; however, they do not possess the above-mentioned properties.

Our contribution. In this paper, we propose a new EDF-based scheduling algorithm to be used for global scheduling of periodic task systems on multiprocessors. We prove that our algorithm successfully schedules any periodic task system with utilization at most $m^2/(2m-1)$ on m identical processors—as $m \to \infty$, this bound approaches $m/2$ from above; hence, it follows that our algorithm successfully schedules any periodic task system with cumulative utilization at most $m/2$ on m identical processors. Our algorithm retains the advantages of EDF—low preemption and interprocessor migration counts; efficient implementation; etc.

[1] Informally, a job becomes *active* at its ready time, and remains so until it has executed for an amount of time equal to its execution requirement, or until its deadline has elapsed.

2. Background

Some very interesting and important results in real-time multiprocessor scheduling theory were obtained in the mid 1990s. We make use of two of these results in this paper; these two results are briefly described below.

2.1. Resource augmentation

Phillips, Stein, Torng, and Wein [14] explored the use of *resource-augmentation* techniques for the on-line scheduling of real-time jobs: the goal was to determine whether an on-line algorithm, if provided with faster processors than those available to a clairvoyant algorithm, could perform better than is possible on same-speed processors.[2] Although we are not studying on-line scheduling in this paper—all the parameters of all the periodic tasks are assumed a priori known—it nevertheless turns out that a particular result from [14] is very useful to us in our study of multiprocessor scheduling. We state this result below.

The focus of [14] was the scheduling of individual jobs, and not periodic tasks. Accordingly, let us define a *job* $J_j = (r_j, e_j, d_j)$ as being characterized by an arrival time r_j, an execution requirement e_j, and a deadline d_j, with the interpretation that this job needs to execute for e_j units over the interval $[r_j, d_j)$.

Let I denote a set of jobs. I is said to be *m-processor feasible on speed-s processors* if I can be scheduled on m processors, each of speed s (i.e., each processor can complete s units of execution per time unit) such that all jobs in I complete by their deadlines.

Theorem 1 (Phillips et al.). *Any instance I that is m-processor feasible on speed-s processors meets all deadlines when scheduled using the earliest deadline first scheduling algorithm* (*EDF*) *on m speed-$(s \cdot (2 - 1/m))$ processors.*

[2] An algorithm is clairvoyant if it has the knowledge of all the future job arrivals; on the other hand, in on-line scheduling, it is assumed that the future job arrivals are not known beforehand. Resource augmentation as a technique for improving the performance of on-line scheduling algorithms was formally proposed by Kalyanasundaram and Pruhs [10].

(Unless otherwise stated, in the rest of the paper, we assume that all processors are of unit speed.)

2.2. Predictable scheduling algorithms

Ha and Liu [8,9,7] have studied the issue of predictability in the multiprocessor scheduling of real-time systems from the following perspective.

Definition 1 (*Predictability*). Let A denote a scheduling algorithm, and $I = \{J_1, J_2, \ldots, J_n\}$ any set of n jobs, $J_j = (r_j, e_j, d_j)$. Let f_j denote the time at which job J_j completes execution when I is scheduled using algorithm A.

Now, consider any set $I' = \{J'_1, J'_2, \ldots, J'_n\}$ of n jobs obtained from I as follows. Job J'_j has an arrival time r_j, an execution requirement $e'_j \leqslant e_j$, and a deadline d_j (i.e., job J'_j has the same arrival time and deadline as J_j, and an execution requirement no larger than J_j's). Let f'_j denote the time at which job J'_j completes execution when I' is scheduled using algorithm A. Scheduling algorithm A is said to be *predictable* if and only if for any set of jobs I and for any such I' obtained from I, it is the case that $f'_j \leqslant f_j$ for all j, $1 \leqslant j \leqslant n$.

Informally, Definition 1 recognizes the fact that the specified execution-requirement parameters of jobs are typically only *upper bounds* on the actual execution-requirements during run-time, rather than the exact values. For a predictable scheduling algorithm, one may determine an upper bound on the completion-times of jobs by analyzing the situation under the assumption that each job executes for an amount equal to the upper bound on its execution requirement; it is guaranteed that the actual completion time of jobs is no later than this determined value.

Since a periodic task system generates a set of jobs, Definition 1 may be extended in a straightforward manner to algorithms for scheduling periodic task systems. An algorithm for scheduling periodic task systems is predictable iff for any periodic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, the job completion time in the case when each job of τ_i has an execution requirement exactly equal to C_i is an upper bound on the completion time of that job when every job of τ_i has an execution requirement of *at most* C_i, for all i, $1 \leqslant i \leqslant n$.

Ha and Liu define a scheduling algorithm to be *priority driven* if and only if it satisfies the condition that *for every pair of jobs J_i and J_j, if J_i has higher priority than J_j at some instant in time, then J_i* always *has higher priority than J_j*. Note that EDF is priority-driven.

The result from the work of Ha and Liu [8,9,7] that we use can be stated as follows.

Theorem 2 (Ha and Liu). *Any priority-driven scheduling algorithm is predictable.*

3. Algorithm EDF-US$[m/(2m-1)]$

In this section, we propose Algorithm EDF-US$[m/(2m-1)]$, a priority-driven scheduling algorithm for scheduling periodic task systems, and derive a utilization-based sufficient feasibility condition for it. In particular, we prove that any periodic task system with utilization at most $m^2/(2m-1)$ can be scheduled by Algorithm EDF-US$[m/(2m-1)]$ to meet all deadlines on m unit-speed processors. We now give a brief overview of our approach. First, we consider a restricted category of periodic task systems, which we call "light" systems and show that the "standard" EDF scheduling algorithm successfully schedules any such system. Then, we apply techniques introduced by Andersson and Jonsson [2] to extend the results concerning light systems to arbitrary systems of periodic tasks. In this process, we modify EDF to obtain Algorithm EDF-US$[m/(2m-1)]$, and derive the feasibility test.

Definition 2. A periodic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is said to be a *light system on m processors* if it satisfies the following two properties.

(P1) $U(\tau) \leqslant \frac{m^2}{2m-1}$.

(P2) For each $\tau_i \in \tau$, $U_i \leqslant \frac{m}{2m-1}$.

Theorem 3. *Any periodic task system τ that is light on m processors is scheduled to meet all deadlines on m processors by EDF.*

Proof. As a direct consequence of properties (P1) and (P2), we can conclude that τ can be scheduled to meet all deadlines on m speed-$(m/(2m-1))$ processors: Pfair and ERfair schedules [3,4,1] and the processor-sharing schedule that assigns a fraction U_i of a processor to τ_i at each time-instant bear witness to this feasibility. Because of the existence of this schedule and the fact that $m/(2m-1) \times (2 - 1/m) = 1$, by Theorem 1, all jobs of τ meet their deadlines when scheduled using EDF on m speed-1 processors. □

Theorem 3 shows that EDF successfully schedules any periodic task system τ with utilization $U(\tau) \leqslant m^2/(2m-1)$ on m identical processors, *provided each $\tau_i \in \tau$ has a utilization $U_i \leqslant m/(2m-1)$*. We now relax the restriction on the utilization of each individual task, i.e., we consider task systems that do not necessarily satisfy property (P2). We first define Algorithm EDF-US$[m/(2m-1)]$ as follows.

Algorithm EDF-US$[m/(2m-1)]$ assigns priorities to jobs of tasks in τ according to the following rule:

if $U_i > m/(2m-1)$ **then** τ_i's jobs are assigned highest priority (ties broken arbitrarily)—this is trivially achieved within an EDF implementation by setting all deadlines of τ_i equal to $-\infty$ (or any negative number, since the deadlines of all the other tasks are positive).

if $U_i \leqslant m/(2m-1)$ **then** τ_i's jobs are assigned priorities according to EDF.

Note that Algorithm EDF-US$[m/(2m-1)]$ is a priority-driven scheduling algorithm. Also note that for $m = 1$, it reduces to plain EDF, since U_i can never be greater than 1.

Theorem 4. *Algorithm* EDF-US$[m/(2m-1)]$ *correctly schedules on m processors any periodic task system τ with utilization $U(\tau) \leqslant m^2/(2m-1)$.*

Proof. Observe that since $U(\tau) \leqslant m^2/(2m-1)$, the number of tasks with utilization greater than $m/(2m-1)$ is less than m. Let k_0 denote the number of such tasks. Let $m_0 \stackrel{\text{def}}{=} m - k_0$.

Let us first analyze the task system $\hat{\tau}$, consisting only of those tasks in τ that have utilization at most $m/(2m-1)$. Therefore, for each $\tau_i \in \hat{\tau}$, we have

$$U_i \leqslant \frac{m}{2m-1} \leqslant \frac{m_0}{2m_0-1}. \tag{1}$$

Furthermore, the total utilization of $\hat{\tau}$ can be bounded from above as follows:

$$\begin{aligned} U(\hat{\tau}) &\leqslant \frac{m^2}{2m-1} - k_0 \cdot \frac{m}{2m-1} \\ &= \frac{m(m-k_0)}{2m-1} \\ &\leqslant \frac{(m-k_0)\cdot(m-k_0)}{2(m-k_0)-1} \\ &= \frac{m_0^2}{2m_0-1}. \end{aligned} \tag{2}$$

From inequalities (1) and (2), we conclude that $\hat{\tau}$ is a periodic task system that is light on m_0 processors. Hence by Theorem 3, $\hat{\tau}$ can be scheduled by Algorithm EDF to meet all deadlines on m_0 processors.

Now, consider the task system $\tilde{\tau}$ obtained from τ by replacing each task $\tau_i \in \tau$ with utilization $U_i > m/(2m-1)$ by a task with the same period, but with utilization equal to one. (This corresponds to making the execution requirement of the task equal to its period.) Note that Algorithm EDF-US$[m/(2m-1)]$ assigns identical priorities to corresponding tasks in τ and $\tilde{\tau}$ (where the notion of "corresponding" is defined in the obvious manner). Also note that when scheduling $\tilde{\tau}$, Algorithm EDF-US$[m/(2m-1)]$ devotes k_0 $(< m)$ processors exclusively to the k_0 tasks that generate jobs of highest priority (since each such task has a utilization equal to unity). The jobs generated by the remaining tasks (the tasks in $\hat{\tau}$) are scheduled using EDF and executed upon the remaining $m_0 = (m-k_0)$ processors. As we have seen above, Algorithm EDF schedules the tasks in $\hat{\tau}$ to meet all deadlines; hence, Algorithm EDF-US$[m/(2m-1)]$ schedules $\tilde{\tau}$ to meet all deadlines of all jobs.

Finally, observe that an execution of Algorithm EDF-US$[m/(2m-1)]$ on task system τ can be considered to be an instantiation of a run of Algorithm EDF-US$[m/(2m-1)]$ on task system $\tilde{\tau}$, in which some jobs—in particular, the ones generated by tasks with utilization one—do not execute to their full execution requirement. Since Algorithm EDF-US$[m/(2m-1)]$ is priority-driven, by Theorem 2, it is a *predictable* scheduling algorithm. Hence every job during the execution of Algorithm EDF-US$[m/(2m-1)]$ on task system τ completes no later than the corresponding job during the execution of Algorithm EDF-US$[m/(2m-1)]$ on task system $\tilde{\tau}$. And thus, from the result in the previous paragraph, we can see that Algorithm EDF-US$[m/(2m-1)]$ correctly schedules τ. □

Can we do better than what is implied by Theorem 4? The exact answer to this question remains open; however, we show below (Theorem 5) that the family of algorithms to which EDF and EDF-US$[m/(2m-1)]$ belong—*priority-driven* scheduling algorithms—cannot in general do much better than the bound of Theorem 4.

Recall that a *priority-driven* scheduling algorithm satisfies the condition that for every pair of jobs J_i and J_j, if J_i has higher priority than J_j at some instant in time, then J_i always has higher priority than J_j. In other words, individual jobs are assigned fixed priorities (although different jobs of the same task may have very different priorities).

Theorem 5. *No priority-driven scheduling algorithm can guarantee correct scheduling of all periodic task systems with utilization at most U on m processors if $U > (m+1)/2$.*

Proof. Consider the periodic task system comprised of $m+1$ identical tasks, each with execution requirement $1+\varepsilon$ and period 2, where ε is an arbitrarily small positive number. Each task releases its first job at time-instant zero. Any priority-driven schedule must assign these jobs fixed priorities relative to each other and the task whose job is assigned the lowest priority at time-instant zero misses its deadline. Note that as $\varepsilon \to 0$, $U(\tau) \to (m+1)/2$; thus, the required result follows. □

4. Conclusions

We have studied the preemptive scheduling of systems of periodic tasks on a platform comprised of several identical multiprocessors. We have proposed Algorithm EDF-US$[m/(2m-1)]$, a new EDF-based multiprocessor algorithm for scheduling periodic task systems. We proved that Algorithm EDF-US$[m/(2m-1)]$ successfully schedules any periodic task system with utilization $\leqslant m^2/(2m-1)$ on m identical processors, while retaining the advantages—low preemption

counts; fewer interprocessor migrations; efficient implementation; etc.—that has made EDF so popular in uniprocessor real-time systems.

References

[1] J. Anderson, A. Srinivasan, Early-release fair scheduling, in: Proc. of the 12th Euromicro Conference on Real-Time Systems, June 2001, pp. 35–43.
[2] B. Andersson, S. Baruah, J. Jonsson, Static-priority scheduling on multiprocessors, in: Proceedings of the Real-Time Systems Symposium, London, December 2001, pp. 193–202.
[3] S. Baruah, N. Cohen, C.G. Plaxton, D. Varvel, Proportionate progress: A notion of fairness in resource allocation, Algorithmica 15 (1996) 600–625.
[4] S. Baruah, J. Gehrke, C.G. Plaxton, Fast scheduling of periodic tasks on multiple resources, in: Proc. of the 9th International Parallel Processing Symposium, April 1995, pp. 280–288.
[5] M. Dertouzos, Control robotics: the procedural control of physical processors, in: Proceedings of the IFIP Congress, 1974, pp. 807–813.
[6] S.K. Dhall, C.L. Liu, On a real-time scheduling problem, Oper. Res. 26 (1) (1978) 127–140.
[7] R. Ha, Validating timing constraints in multiprocessor and distributed systems, PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, IL, 1995; Available as Technical Report No. UIUCDCS-R-95-1907.
[8] R. Ha, J.W.S. Liu, Validating timing constraints in multiprocessor and distributed real-time systems, Tech. Rept. UIUCDCS-R-93-1833, Department of Computer Science, University of Illinois at Urbana-Champaign, IL, October 1993.
[9] R. Ha, J.W.S. Liu, Validating timing constraints in multiprocessor and distributed real-time systems, in: Proceedings of the 14th IEEE International Conference on Distributed Computing Systems, IEEE Computer Society Press, Los Alamitos, CA, June 1994.
[10] B. Kalyanasundaram, K. Pruhs, Speed is as powerful as clairvoyance, in: 36th Annual Symposium on Foundations of Computer Science (FOCS'95), IEEE Computer Society Press, Los Alamitos, CA, October 1995, pp. 214–223.
[11] C.L. Liu, Scheduling algorithms for multiprocessors in a hard real-time environment, JPL Space Programs Summary 37–60 (II) (1969) 28–31.
[12] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, J. ACM 20 (1) (1973) 46–61.
[13] A. Mok, Task management techniques for enforcing ED scheduling on a periodic task set, in: Proceedings of the 5th IEEE Workshop on Real-Time Software and Operating Systems, Washington, DC, May 1988, pp. 42–46.
[14] C.A. Phillips, C. Stein, E. Torng, J. Wein, Optimal time-critical scheduling via resource augmentation, in: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 4–6 May 1997, pp. 140–149.