Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 77 (2005) 67–80
The Journal of Systems and Software
www.elsevier.com/locate/jss

Fair scheduling of dynamic task systems on multiprocessors ☆

Anand Srinivasan, James H. Anderson *

Department of Computer Science, University of North Carolina, 256, Sitterson Hill, CB #3175, Chapel Hill, NC 27599-3175, USA

Received 8 November 2002; received in revised form 1 September 2003; accepted 12 December 2003
Available online 11 September 2004

Abstract

In dynamic real-time task systems, tasks that are subject to deadlines are allowed to join and leave the system. In previous work, Stoica et al. and Baruah et al. presented conditions under which such joins and leaves may occur in fair-scheduled uniprocessor systems without causing missed deadlines. In this paper, we extend their work by considering fair-scheduled multiprocessors. We show that their conditions are sufficient on M processors, under any deadline-based Pfair scheduling algorithm, if the utilization of every subset of $M-1$ tasks is at most one. Further, for the general case in which task utilizations are not restricted in this way, we derive sufficient join/leave conditions for the PD^2 Pfair algorithm. We also show that, in general, these conditions cannot be improved upon without causing missed deadlines.

Keywords: Dynamic task systems; Pfairness; Multiprocessor; Real-time scheduling

1. Introduction

In many real-time systems, the set of runnable tasks may change dynamically. For example, in an embedded system, different modes of operation may need to be supported; a mode change may require adding new tasks and deleting existing tasks. Another example is a desktop system that supports real-time applications such as multimedia and collaborative-support systems, which may be initiated at arbitrary times.

The distinguishing characteristic of *dynamic* task systems such as these is that tasks are allowed to *join* and *leave* the system. If such joins and leaves are unrestricted, then the system may become overloaded, and deadlines may be missed. Thus, joins and leaves must be performed only under conditions that ensure that deadline guarantees are not compromised. A suitable join condition usually can be obtained from the feasibility test associated with the scheduling algorithm being used. A leave condition is somewhat trickier. In particular, if an "over-allocated" task is allowed to leave, then it might re-join immediately, and thus effectively execute at a higher-than-prescribed rate.

In this paper, we consider the problem of scheduling such task systems on multiprocessors. This problem has been studied earlier in the context of uniprocessor static-priority (Sha et al., 1989; Tindell et al., 1992) and fair-allocation schemes (Baruah et al., 1998; Stoica et al., 1996). Our focus is fair scheduling because it is the only known way of optimally scheduling recurrent real-time tasks on multiprocessors (Anderson and Srinivasan, 2004; Baruah et al., 1996; Baruah et al., 1995; Srinivasan and Anderson, 2002). In addition, practical interest in multiprocessor fair scheduling algorithms is growing. For example, Ensim Corp., an Internet service provider, has deployed such algorithms in its product line (Keshav, 2001). The need to support dynamic tasks is fundamental in this setting.

☆ Work supported by NSF grants CCR 9972211, CCR 9988327, ITR 0082866, and CCR 0204312.
* Corresponding author.
E-mail address: anderson@cs.unc.edu (J.H. Anderson).

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2003.12.041

In fair scheduling disciplines, tasks are required to make progress at steady rates. Steady allocation rates are ensured by scheduling in a manner that closely tracks an ideal, fluid allocation. The *lag* of a task measures the difference between its ideal and actual allocations. In fair scheduling schemes, lags are required to remain bounded. (Such a bound in turn implies a bound on the timeliness of real-time tasks.) If a task's lag is positive, then it has been under-allocated; if negative, then it has been over-allocated. In the uniprocessor join/leave conditions presented previously (Baruah et al., 1998; Stoica et al., 1996, a task is allowed to leave iff it is not over-allocated, and join iff the total utilization after it joins is at most one.

Extending the above-mentioned work to multiprocessors is not straightforward; in fact, Baruah et al. explicitly noted the multiprocessor case as an open problem Baruah et al., 1998. In recent work, dynamic multiprocessor systems were considered by Chandra et al. (2000, 2001). However, their work was entirely experimental in nature, with no formal analysis of the algorithms considered. In this paper, we present join/leave conditions for which such analysis is provided. Before presenting a more detailed overview of the contributions of this paper, we briefly describe some of the fair scheduling concepts used in this paper.

Pfair scheduling. The *periodic* task model provides the simplest notion of a recurrent real-time task. In this model, successive job releases (i.e., invocations) by the same task are spaced apart by a fixed interval, called the task's *period*. Periodic tasks can be optimally scheduled on multiprocessors using *Pfair* scheduling algorithms Anderson and Srinivasan, 2004; Baruah et al., 1996; Baruah et al., 1995. Pfairness requires the lag of each task to be bounded between -1 and 1, which is a stronger requirement than periodicity. As we shall see, these lag bounds have the effect of breaking each task into quantum-length *subtasks* that must be scheduled within *windows* of approximately equal lengths. The length and alignment of a task's windows are determined by its weight. The *weight* or *utilization* of a task is the ratio of its per-job execution cost and period; a task's weight determines the processor share it requires. Fig. 1(a) shows the subtasks and windows for the first two jobs of a periodic task T with an execution requirement of 8 and a period of 11 (i.e., of weight 8/11).

In the *sporadic* model, the periodic notion of recurrence is relaxed by specifying a minimum (rather than exact) spacing between consecutive job releases of the same task. In recent work (Anderson and Srinivasan, 2000; Srinivasan and Anderson, 2002), we extended the sporadic model to obtain the *intra-sporadic* (IS) and *generalized intra-sporadic* (GIS) models. The sporadic model allows *jobs* to be released "late"; the IS model allows *subtasks* to be released late, as illustrated in Fig. 1(b). The GIS model is obtained from the IS model by allowing subtasks to be absent. Fig. 1(c) shows an example.

In Srinivasan and Anderson (2002), we showed that the PD^2 Pfair algorithm optimally schedules static GIS task systems on multiprocessors. In Anderson and Srinivasan (2000), we proved that the (simpler) earliest-pseudo-deadline-first (EPDF) algorithm is optimal for scheduling static IS task systems on two processors. (PD^2 and EPDF are described in Section 2.2.)

Contributions. In this paper, we extend our earlier work, as well as prior work on uniprocessor fairness, in several significant ways. First, we show that the previously-presented uniprocessor join/leave conditions (Baruah et al., 1998; Stoica et al., 1996) are insufficient for avoiding deadline misses when tasks are scheduled using any of a class of algorithms that includes all known (dynamic-priority) Pfair scheduling algorithms. Second, we show that these uniprocessor conditions are sufficient when using any deadline-based algorithm, if the total weight of any subset of $M-1$ tasks is at most one at all times. This result extends our earlier result on the optimality of EPDF for two-processor systems (Anderson and Srinivasan, 2000). Third, we derive sufficient conditions (that are tight) for the general case in which task weights are not restricted as above, and PD^2 is used for scheduling.

Overview. The rest of the paper is organized as follows. In Section 2, needed definitions are given. In Section 3, our join/leave conditions are stated. Results pertaining to the EPDF and PD^2 algorithms are then presented in Sections 4 and 5, respectively. We conclude in Section 6.

2. Preliminaries

In the following subsections, relevant concepts and terms are defined. We begin with Pfair scheduling.

2.1. Pfair scheduling

In defining notions relevant to Pfair scheduling, we limit attention (for now) to periodic tasks; we assume that each such task releases its first job at time 0. A periodic task T with an integer *period* $T.p$ and an integer per-job *execution cost* $T.e$ has a *weight* $wt(T) = T.e/T.p$, where $0 < wt(T) \leqslant 1$. Such a task T is *light* if $wt(T) < 1/2$, and *heavy* otherwise.

Under Pfair scheduling, processor time is allocated in discrete time units, called *quanta*; the time interval $[t, t+1)$, where t is a nonnegative integer, is called *slot* t. (Hence, time t refers to the beginning of slot t.) In each slot, each processor can be allocated to at most one task. A task may be allocated time on different processors, but not in the same slot (i.e., interprocessor migration is allowed but parallelism is not). The sequence of allo-