Available online at www.sciencedirect.com
SCIENCE DIRECT®

J. Parallel Distrib. Comput. 66 (2006) 47–67

Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Supporting lock-free synchronization in Pfair-scheduled real-time systems☆

Philip Holman, James H. Anderson*

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

Received 16 August 2004; received in revised form 7 June 2005; accepted 10 June 2005
Available online 18 August 2005

Abstract

We consider the use of lock-free techniques for implementing shared objects in real-time Pfair-scheduled multiprocessor systems. Lock-free objects are more economical than locking techniques when implementing relatively simple objects such as buffers, stacks, queues, and lists. However, the use of such objects on real-time multiprocessors is generally considered impractical due to the need for analytical real-time guarantees. In this paper, we explain how the quantum-based nature of Pfair scheduling enables the effective use of such objects on real-time multiprocessors and present analysis specific to Pfair-scheduled systems. In addition, we show that analytical improvements can be obtained by using such objects in conjunction with group-based scheduling techniques. In this approach, a group of tasks is scheduled as a single entity (called a *supertask* in the Pfair literature). Such grouping prevents tasks from executing simultaneously, and hence from executing in parallel. Consequently, grouping tasks can improve the worst-case scenario with respect to object contention. Grouping also enables the use of less costly *uniprocessor* algorithms when all tasks sharing an object reside within the same group. We illustrate these optimizations with a case study that focuses on shared queues. Finally, we present and experimentally evaluate a simple heuristic for grouping tasks in order to reduce object contention. Though the analysis presented herein focuses specifically on Pfair-scheduled systems, the observations and techniques should be applicable to other quantum-scheduled systems as well.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Lock-free; Synchronization; Pfairness; Real-time; Scheduling; Multiprocessor; Supertasking

1. Introduction

There has been much recent work on scheduling techniques that ensure fairness, temporal isolation, and timeliness among *real-time* tasks multiplexed on a set of processors. Real-time tasks differ from conventional processes in that they must satisfy timing constraints in order to ensure correct operation. The most common form of constraint is the use of *per-job deadlines*. Under task models that use this type of constraint, each task is invoked repeatedly, and each such invocation is called a *job* of the task. Furthermore, each job must complete execution within a fixed amount of time, called its *relative deadline*.

Real-time systems are often classified by their strictness. Systems in which it is desirable, but not necessary, to meet all constraints are called *soft* real-time systems. For example, a video decoder usually has soft real-time constraints: timing violations may be acceptable, provided they occur rarely. Techniques for verifying the correctness of such systems vary significantly, often depending on their intended use. Systems in which it is unacceptable to violate any constraints are called *hard* real-time systems. For instance, control systems used to coordinate the operation of mechanical parts on assembly lines often require strict constraints. For hard real-time systems, constraints are guaranteed by subjecting the systems to *worst-case* analysis.

☆ Work supported by NSF grants CCR 9972211, CCR 9988327, ITR 0082866, CCR 0204312, and CCR 0309825. Some results in this paper were presented in preliminary form at the 14th Euromicro Conference on Real-time Systems [17].

* Corresponding author. Fax: +919 962 1799.
E-mail address: anderson@cs.unc.edu (J.H. Anderson).

Worst-case analysis consists of analyzing scenarios that are conservatively worse than any situation that can actually arise under normal operation, i.e., if constraints cannot be violated under a "worse-case" scenario, then they cannot possibly be violated under any realistic scenario. For instance, one common technique used to construct a worst-case scenario is to assume that each job in the system consumes the maximum amount of processor time possible, called the job's *worst-case execution time*. In the real system, it is highly unlikely, if not impossible, that all jobs will exhibit this property. However, such an assumption guarantees that the constructed scenario is a conservative approximation of any real scenario.

Due to the practical differences between conventional and hard real-time systems, the philosophies underlying the designs of these systems are also very different. In a conventional system, *average-case* performance is paramount, and the quality of a system is typically determined through *measurement*. The use of heuristics at runtime is common and designs are often very complex. Hard real-time systems, on the other hand, require *predictability* most of all. Complex designs and the online use of heuristics both reduce predictability, which results in overly conservative worst-case scenarios. Such overestimation is undesirable because it leads to chronically underutilized resources. In addition, the quality of a system is determined *analytically* rather than empirically, i.e., by determining how far the system can be stressed before timing constraints can no longer be guaranteed. Empirical testing is almost always insufficient for this purpose due to the need to focus on worst-case situations.

1.1. Fair scheduling of real-time multiprocessors

In recent years, there has been considerable interest in fair scheduling algorithms for real-time multiprocessor systems [3–5,7,8,10]. Under *fair* scheduling disciplines, tasks are assigned weights, and each task is granted processor time in proportion to its weight. Allocating processor time in this fashion provides two advantages. First, the amount of processor time allocated to a task is not affected by "misbehaving" tasks, i.e., tasks that attempt to consume more processor time than their specified maximum. Second, when using a fixed number of processors, proportional allocation of processor time implies bounds on the amount of processor time allocated to each task. These bounds can then be used with worst-case analysis to provide hard real-time guarantees.

This latter advantage was demonstrated by Baruah et al. in their work on the *proportionate-fairness* (*Pfair*) scheduling approach [7,8]. This approach distinguishes itself by being the only currently known technique for optimally scheduling hard real-time multiprocessor systems. In a series of papers [3–5], Anderson and Srinivasan further demonstrated the utility of fair scheduling by proposing extensions to Pfair scheduling that support *rate-based* execution of tasks. Under rate-based scheduling, the execution rate of tasks is permitted to vary over time without impacting the system's hard real-time guarantees. (In the work of Baruah et al., all tasks are assumed to execute at steady rates at all times.) In [10], Chandra et al. investigated the use of an alternative fair scheduling approach, called *surplus fair* scheduling, to schedule multimedia applications. Though suboptimal, such alternative approaches often have practical advantages that make them better-suited to soft real-time and conventional systems.

1.2. Research focus

One limitation common to prior work on real-time multiprocessor fair scheduling algorithms is that only *independent* tasks that do not synchronize or share resources have been considered. In contrast, tasks in real systems usually are not independent. Synchronization entails additional overhead, which must be taken into account when performing real-time analysis (e.g., see [2,6,26]). Unfortunately, prior work on synchronization in real-time systems has been directed at uniprocessor systems [2], or systems implemented using non-fair scheduling algorithms [26] (or both [6]), and thus cannot be directly applied in fair-scheduled real-time multiprocessor systems. [1]

The goal of our research is to design an efficient hard real-time operating system around the Pfair scheduling approach and its variants [3–5,7,8]. A necessary step toward this goal is the development of synchronization mechanisms and analysis to support their use. In this paper, we explain how to incorporate lock-free objects into such a system. The importance of our results lies in the fact that lock-free synchronization is typically considered impractical in multiprocessor real-time systems. As we later explain, the efficient use of lock-free objects is enabled by the structure provided by Pfair scheduling. We also show that several straightforward techniques exist for *reducing* the overhead of lock-free synchronization on multiprocessors. Though the presented analysis applies only to Pfair-scheduled systems, the observations and concepts upon which the results are based can be applied to other multiprocessor real-time systems as well.

1.3. The synchronization problem

In real-time systems, timing violations can result from contention for a shared resource. The most common approach to synchronizing access to a shared resource is the use of *semaphore-based locking*. Under this approach, a job must request and obtain the lock associated with a resource before using the resource. Once the resource access is complete, the lock is relinquished. A region of code guarded by a lock is called a *critical section*. In real-time systems, the

[1] Indeed, mechanisms for supporting synchronization in fair-scheduled real-time *uniprocessor* systems were first considered only very recently, e.g., in [9,12,22].