Some Guidelines for Proportional Share CPU Scheduling in General-Purpose Operating Systems

John Regehr
School of Computing
University of Utah
regehr@cs.utah.edu

Abstract

Our premise is that since there already exists a large, mature body of literature on real-time scheduling in general-purpose operating systems, it is time to spend more effort deciding which of these algorithms should be used and when, and less effort on generating new algorithms. In this paper we focus on proportional share schedulers. We introduce the notion of pessimism*—the proportion of over-reservation required for an application to meet real-time deadlines when scheduled by proportional share schedulers that have bounded allocation error. We study the implications of pessimism and its effect on the selection of scheduling algorithm and scheduling quantum size, and also the interaction of quantum size and context switch overhead. Finally, we examine the implications of these tradeoffs for the design of applications and schedulers.*

1. Introduction

There are compelling reasons to use proportional share scheduling techniques to support multimedia and other soft real-time applications on general-purpose operating systems. First, proportional share (PS) schedulers are a good match for existing infrastructure such as a periodic timer interrupt and mechanisms for assigning priorities to applications—priorities can be mapped to shares in a proportional-share environment. Second, PS schedulers provide stronger guarantees to applications than do traditional time-sharing schedulers: they allocate a specific fraction of the CPU to each thread, and some schedulers provide error bounds on the allocation rate. Third, PS schedulers have clear semantics during underload: excess CPU time is allocated fairly, in contrast with some reservation-based schedulers that must idle or back off to a secondary scheduling policy once all application budgets are exhausted.

2. Fairness and Allocation Error

An important property of PS scheduling techniques is that they characterize threads with a single parameter, a share, and do not make a distinction between the rate at which a thread requires CPU time and the granularity at which it must receive that rate. Consequently, PS schedulers are often primarily evaluated based on the level of fairness that they can provide. Secondary evaluation criteria include implementation complexity and run-time efficiency. Schedulers such as EEVDF [8] are *optimally fair* in the sense that a thread's allocation, measured over any time interval, never deviates more than a single time quantum from its ideal (infinitely fine-grained) allocation. This is provably the smallest allocation error that a PS scheduler can provide; other PS schedulers, such as start-time fair queuing [3], have larger error bounds. So, although PS schedulers may bound allocation error, the bound is a function of the scheduler and the quantum size—threads being scheduled don't get to choose it directly.

The predominant model for real-time task execution [5] requires that each periodic task receive a certain amount of CPU time during each period. This model can be straightforwardly applied to the many multimedia and other soft real-time applications that are becoming popular on general-purpose operating systems. These include voice recognition, software signal processing [4], and the presentation of audio and video, which may be stored or live, and local or remote. Mapping PS guarantees into the periodic task model requires taking the error bound into account. If a task with worst-case execution time (WCET) c and period p receives share s of the CPU from a PS scheduler with error bound d then, as Stoica et al. [7] have observed, the inequality

$$sp - d > c \tag{1}$$

must hold if the task is to be guaranteed to meet all deadlines.

We define the pessimism P of a particular PS guarantee to be sp/c, the amount of CPU time reserved for a thread divided by the amount of time it actually requires. The ideal level of pessimism, 1.0, can only be achieved by a guarantee satisfying the above inequality when d is zero. If the pessimism of a guarantee is 1.5, then 50% of the CPU time allocated to the associated thread is being wasted in the sense that it cannot be guaranteed to another thread.

3. Managing Pessimism

Since the period p and execution time c are fixed for a given real-time application and reserved share s is not an independent variable, the available mechanism for reducing pessimism is to minimize the allocation error d. There are two ways to do this. First, we can choose a scheduling algorithm with a low error bound, such as EEVDF. For purposes of this paper we assume that EEVDF or an equivalent optimal algorithm is used, and consequently that the allocation error bound is the same as the quantum size. Second, we can reduce the size of a scheduling quantum. A typical value for general-purpose operating systems is 10–30 ms, but quantum size can be reduced almost arbitrarily. In practice, the desire for small quanta must be balanced with the associated overheads of periodic timer interrupts and context switches. Since a very fair scheduler like EEVDF will usually cause a context switch at every timer interrupt, from this point forward we will only consider the cost of context switches, which dominate overall cost.

If we turn the inequality in Equation 1 into an equality and solve for share, we get $s = (c + d)/p$. Substituting this into the formula for pessimism, we get $P = ((c+d)/p)(p/c)$. This simplifies to $P = (c+d)/c$, implying that the pessimism of an application's guarantee does not depend on its period, but only on the allocation error and the amount of time it requires per period. Figure 1 shows pessimism as a function of scheduling quantum size for three application WCETs: 3 ms, 10 ms, and 30 ms. Pessimism is bounded below by 1.0, and we clip the top of the graph at 4.0 under the assumption that guaranteeing an application more than four times the CPU that it actually requires is completely unacceptable.

Figure 1 shows that for applications with small processing requirements, such as an audio player that requires 3 ms of CPU time every 50 ms in order to decode data and hand it to a sound device, even fairly short scheduling quanta imply a large degree of pessimism in guarantees. On the other hand, for real-time applications with large processing requirements per period, such as a vision algorithm that requires 30 ms of CPU time during every 100 ms period in order to process 10 frames per second, quantum sizes in the 10–30 ms range result in levels of pessimism that are not too large.

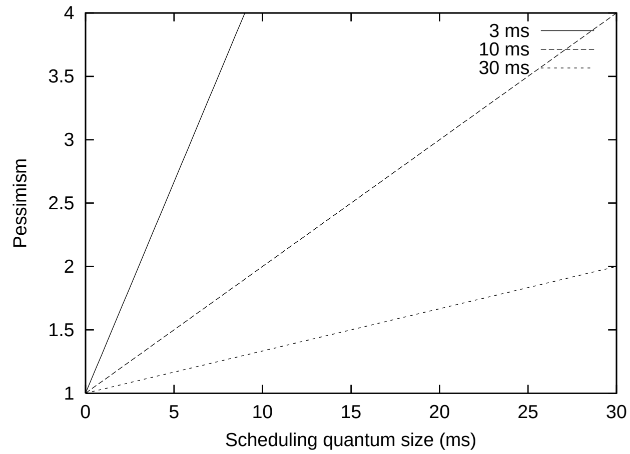

Figure 1. Pessimism as a function of quantum size

4. Managing Context Switch Overhead

In the previous section we showed that it is possible to minimize the amount of pessimism required to make real-time guarantees to applications using PS schedulers if the scheduler quantum size can be made to be small. In this section, we explore the consequences of reducing the quantum size.

By preempting a thread at the end of each quantum, a scheduler incurs a certain amount of overhead. For example, on a 500 MHz Pentium III running Windows 2000, we have measured the median cost of executing the path through the kernel context switch code at 7.1 μs. However, the actual cost of a context switch can easily exceed this nominal cost because a newly scheduled application will often be running on a cold cache, and will be forced to wait on memory reads as its working set is paged into the level 2 cache. In previous work [6, Ch. 10] we have shown that on the platform described above, which has 512 KB of L2 cache memory, the actual cost of a context switch is an order of magnitude larger than the nominal cost for applications with a working set of 8 KB, and two orders of magnitude larger for applications with a working set of 79 KB. The cost of a context switch continues to increase with working set size, up to a maximum of about 2.5 ms for working sets near 512 KB.

Figure 2 shows the amount of overhead caused by context switches for different scheduling quantum sizes, assuming that a context switch occurs at each quantum boundary. The lowest line—corresponding to the nominal 7.1 μs cost of the kernel context switch code path—indicates that this is not a significant source of overhead, even for quantum sizes well below 5 ms. The other lines, however, show

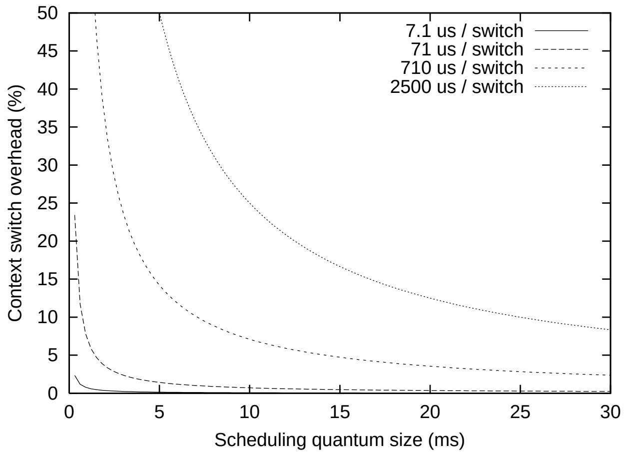

Figure 2. Context switch overhead as a function of quantum size

that for applications that make good use of the cache, even quantum sizes in the 10–30 ms range can cause significant overhead. It is therefore necessary to exercise caution when reducing quantum size.

5. Implications for Application Design

Although the period, and the amount of CPU time required per period, are inherent properties of a real-time application, in some cases good design can reduce dependencies on precise scheduling behavior from the operating system. For example, a video player can buffer a number of decoded frames rather than decoding each frame just in time. This increases the effective period of the decoder, ensuring that short-term allocation error will not impact playback quality.

A video player that must produce 33 frames per second has a period of 30 ms. However, if it buffers 10 decoded frames, its effective period becomes 300 ms. Assuming that 10 ms of CPU time are required to decode each frame, when scheduled using EEVDF on a machine with a scheduling quantum 5 ms long the unbuffered video player must be scheduled with $s = 0.5$ to be guaranteed to not drop any frames. This reservation has a pessimism value of 3. The same player, when configured to buffer 10 frames, must be scheduled with $s = 0.35$, for a pessimism value of 1.05—a significant reduction.

6. Implications for Scheduler Design

Since PS schedulers are not aware of application deadlines, they can be too fair. Abeni et al. [2] have shown that constant bandwidth allocation (CBA) does not have this problem. The strength and weakness of CBA and similar CPU reservation mechanisms is that they explicitly schedule applications to meet deadlines. By making a distinction between the amount of CPU time that an application requires and the granularity at which that amount is delivered, they can greatly reduce the number of unnecessary context switches. On the other hand, they require that applications specify their periods as well as worst-case execution times per period, placing a larger burden on developers. However, the period of an application is often much easier to determine than the WCET, which is strongly platform dependent and may depend on data as well. In fact, tasks often implicitly inform the operating system of their periods by using a timer service to be awakened at the beginning of each period.

Limiting the fairness of scheduling for applications that do not require it can be regarded as a throughput optimization. A further optimization is to entirely eliminate periodic timer interrupts, replacing them with a precisely settable interrupt. This is useless on systems with PS schedulers and fixed-size scheduling quanta, but can be used to provide low-jitter, precisely enforced CPU allocation on systems that support CPU reservations.

The properties of some PS schedulers have been worked out in the context of variable-sized scheduling quanta. Although allowing applications to request different quantum sizes permits a decoupling between scheduling rate and granularity, it also sacrifices two of the main benefits of PS schedulers—that they require only one parameter per thread, and that they can reuse the existing periodic timer interrupt facility that general-purpose operating systems use to make thread preemption decisions. In fact, for a fixed set of CPU-bound applications, we believe that the schedules produced by a proportional share scheduler with variable quantum size and a work-conserving CPU reservation algorithm such as the constant bandwidth server are the same [1].

7. Conclusions

This work is part of our ongoing research in real-time scheduling techniques for general-purpose operating systems. This domain is very demanding in the sense that the real-time subsystem must not complicate the programming model seen by application developers, and it must not significantly impact operating system throughput for any important workload.

We have introduced the notion of *pessimism*—the proportion of over-reservation required for an application to meet real-time deadlines when scheduled by proportional share schedulers that have bounded allocation error. We have studied the implications of pessimism and its effect on the selection of scheduling quantum size, and also the interaction of quantum size and context switch overhead. Finally, we have examined the implications of these tradeoffs for the design of applications and schedulers.

We conclude that there exists a false dichotomy between schedulers based on proportional share techniques and schedulers based on the Liu and Layland task model. The important question is not which class of algorithms is better, but rather, for a given operating system and set of applications, (1) to what degree must existing infrastructure such as a periodic timer interrupt and system for manipulating priorities be utilized; (2) how much pessimism and context switch overhead is acceptable; and, (3) what scheduling parameters can the developers of real-time applications be reasonably expected to provide?

References

[1] Luca Abeni. Personal communication, June 2001.

[2] Luca Abeni, Giuseppe Lipari, and Giorgio Buttazzo. Constant bandwidth vs proportional share resource allocation. In *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, June 1999.

[3] Pawan Goyal, Xingang Guo, and Harrick M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *Proc. of the 2nd Symposium on Operating Systems Design and Implementation*, pages 107–121, Seattle, WA, October 1996.

[4] Michael B. Jones and Stefan Saroiu. Predictable scheduling for a soft modem. Technical Report MSR-TR-2000-88, Microsoft Research, December 2000.

[5] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[6] John Regehr. *Using Hierarchical Scheduling to Support Soft Real-Time Applications on General-Purpose Operating Systems*. PhD thesis, University of Virginia, May 2001.

[7] Ion Stoica, Hussein Abdel-Wahab, and Kevin Jeffay. On the duality between resource reservation and proportional share resource allocation. In *Proc. of Multimedia Computing and Networking 1997*, pages 207–214, San Jose, CA, February 1997.

[8] Ion Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and C. Greg Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proc. of the 17th IEEE Real-Time Systems Symposium*, pages 288–299, Washington DC, December 1996.