In *Proceedings of the International Workshop on Multimedia Middleware (M3W '01)*, pages 23–27, Ottawa, Canada, October 2001.

The Case for Using Middleware to Manage Diverse Soft Real-Time Schedulers

John Regehr Jay Lepreau
School of Computing, University of Utah
50 S. Central Campus Drive, Room 3190
Salt Lake City, UT 84112–9205, USA
{regehr, lepreau}@cs.utah.edu

ABSTRACT

Although a number of general-purpose operating systems have been extended with soft real-time schedulers and have the potential to support coexisting, independently developed real-time applications, this potential is currently largely unexploited by common applications. This is because the provided scheduling functionality is low-level and depends on parameters that are difficult to estimate, and because different semantics are provided by different schedulers. The cost/benefit ratio of real-time support in general-purpose operating systems is too high for most users and application developers to tolerate.

The contribution of this paper is the design of the CPU Resource Manager (CRM): a middleware application that manages processor allocation in a QoS-enabled general-purpose operating system by (1) providing a level of indirection between applications and the scheduling subsystem, (2) automatically calculating scheduling parameters when applicable, and (3) providing an environment supporting the execution of user-specified rules about the allocation of processor time. The focus of this work is not to increase the benefit provided by real-time schedulers, but rather to decrease the cost of using them.

Keywords

Middleware, multimedia, open systems, resource management, scheduler composition, scheduler parameter estimation, soft real-time.

1. INTRODUCTION

Scheduling subsystems that have been added to general-purpose operating systems are more than adequate to meet the fairly modest predictability requirements of coexisting, independently developed soft real-time applications. For example, Eclipse [2] extends FreeBSD, Rialto/NT [6] and HLS [10] extend Windows 2000, and there are at least five publicly available systems providing soft real-time scheduling in Linux [4, 9, 14, 17, 18].

There are several obstacles to widespread use of these systems. First, application developers are unlikely to support scheduling abstractions that (at first) are present on only a small number of machines. Second, these different schedulers provide different scheduling abstractions, making it difficult to match these abstractions with application requirements. Third, real-time schedulers require parameters such as share or worst-case execution time that are effectively impossible to determine in advance and may even be difficult to accurately estimate at run time.

This paper outlines the design of the CPU Resource Manager (CRM), which represents a step towards surmounting these obstacles. We plan to implement CRM in Linux because it has a wide variety of readily available real-time support, and to eventually port it to other operating systems. Our thesis is that the common functionality between existing soft real-time scheduling abstractions can be exploited by QoS-aware middleware to hide the semantic differences between scheduling abstractions. This makes the choice of underlying scheduling abstraction irrelevant to users and applications as long as basic services such as load isolation and bounded-latency scheduling are provided. In addition, a middleware QoS manager provides a convenient mechanism for implementing user-specified rules about processor allocation, estimating the scheduling parameters of applications and automatically applying them when applications are started, and storing these parameters and other information such as user-specified application importances. The goal of this work is to unobtrusively allow application developers and end-users to benefit from QoS-enabled operating systems.

In the next section we give an overview of CRM. Sections 3–5 describe our approach to dealing with the specific research challenges that must be solved in order to implement this software. In Section 6 we compare our approach to the related work and we conclude in Section 7.

2. CRM GOALS AND DESIGN

In previous work [11] we identified the application programming model as a key issue in the design of operating system support for soft real-time applications. We identified

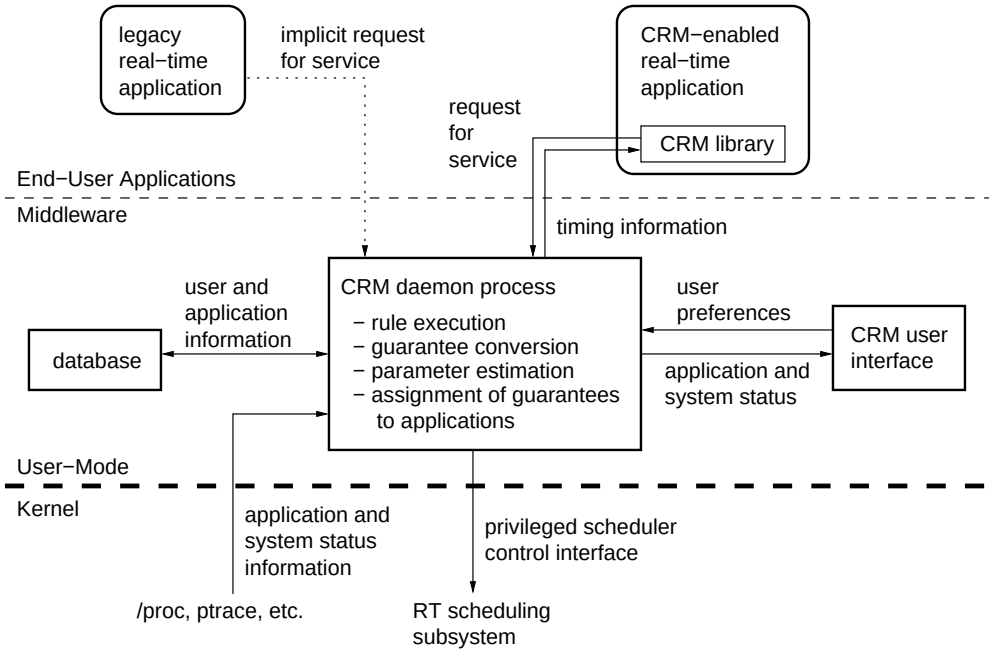

Figure 1: Structure of CRM and relationships with other system components

a maxim that states that application developers will ignore system support for real-time if it increases the difficulty of their tasks too much. Rather, they will assume away the problem of scheduling contention, forcing their users to make this assumption true by not running several real-time applications concurrently.

A resource manager for CPU time must be unobtrusive: its default mode of operation should be to add value behind the scenes by estimating application scheduling requirements and automatically applying them to applications as they are started. During overload, the resource manager should have a sensible default behavior such as reducing the CPU time available to some applications, as opposed to applying admission control and refusing to run a new application. Although it will avoid the frustrating and confusing (to users) behavior of refusing to start running a new application, this heuristic will sometimes cause the performance of running applications to be degraded. However, it can be refined by presenting the user with a simple graphical interface that permits selected applications to be identified as important, making them exempt from being "squeezed" in the future. Or, more likely, the resource manager would be distributed with a default list of applications such as CD burners and audio players that are known to provide little value when given less than their full CPU requirement.

The structure of our CPU Resource Manager, CRM, is shown in Figure 1. Although CRM will be structured as a suite of tools and libraries, its main component is a user-level daemon that executes user-specified rules about processor allocation, performs conversions between different kinds of guarantees, estimates the parameters for scheduling guarantees that should be provided to applications, and assigns appropriate scheduling behavior to applications. The CRM daemon interacts with a database that manages a persistent store of application and user information, a graphical user interface, kernel scheduling and application monitoring subsystems, and library code that can be linked into applications to provide them with instrumentation and fine-grained control over scheduling behavior.

The performance overhead of CRM will be low because it is only active when an application starts, terminates, or has a change of requirements. In other words, CRM controls medium- and long-term CPU allocation, leaving fine-grained scheduling decisions to the operating system thread scheduler.

The next three sections describe the techniques that we believe will make CRM possible.

3. CONVERTING BETWEEN SOFT REAL-TIME GUARANTEES

This section provides technical background for the conversion between types of real-time scheduling. In previous work [10] (some of which is currently in submission [12]) we developed these conversions to make it possible to reason about the properties provided by CPU schedulers that are composed in a hierarchy.

The function of a real-time scheduler can be viewed as providing *guarantees* to applications about the distribution of CPU time that they will receive. For example, a scheduler that provides CPU reservations might guarantee an application to receive 5 ms of CPU time during any 33 ms time interval for as long as the guarantee remains in effect. Since all real-time guarantees bound CPU allocation during a period of time, it is sometimes possible to convert between these

guarantees. In this paper we are concerned only with schedulers that, unlike fixed-priority and time-sharing schedulers, *enforce* limits on application CPU usage. We now briefly characterize several guarantees provided by soft real-time schedulers that have been described in the literature.

CPU Reservations have a *period* and an *amount* (with the application being guaranteed to receive the amount of CPU time during each period of time). CPU reservations may be characterized as *basic* or *continuous* and as *hard* or *soft*. These properties are orthogonal. The hard versus soft distinction (terminology that we borrow from Oikawa and Rajkumar [9]) does not reflect a difference between hard and soft real-time, but rather the fact that hard CPU reservations guarantee upper and lower bounds on the allocation of CPU time while soft reservations guarantee only a lower bound. Therefore, hard reservations are useful for limiting the rate at which applications run while soft reservations are useful for applications that require a minimum amount of CPU time to operate correctly but can take advantage of extra time to provide added value.

Continuous CPU reservations guarantee that a thread will receive a certain amount of CPU time during *any* time interval of a certain size, while basic reservations guarantee that a thread will receive an amount of processor time during time intervals of application-specified duration, with the interval boundaries chosen by the scheduler. The distinction can be understood by observing that basic reservation schedulers have the freedom to rearrange CPU time within a period (for example, scheduling an application at the beginning of one period and the end of the next), while continuous reservation schedulers do not have this freedom. The portable Resource Kernel [9] provides hard and soft basic CPU reservations. Rialto [7] and Rialto/NT [6] provide soft, continuous reservations.

Proportional Share schedulers guarantee an application with s shares to receive s/t of the CPU where t is the total number of shares over all applications. However, all proportional share schedulers are quantum-based and introduce error with respect to this ideal model of CPU allocation. Some of these schedulers do not bound allocation error and are not real-time schedulers. A guarantee provided by a scheduler that bounds allocation error is characterized by a parameter δ that indicates the largest possible difference between the actual amount of CPU time that a thread may receive during an arbitrary time interval and the thread's ideal share of the CPU during the same interval. For example, the EEVDF scheduler [16] bounds allocation error to the size of a single scheduling quantum, while the error for start-time fair queuing [17] is a function of a thread's share, the quantum size, and the number of threads being scheduled.

Soft real-time guarantees can be *converted* into other guarantees. This means that scheduling behavior that meets the criteria for one kind of guarantee also provably meets the criteria for being another kind of guarantee. Some conversions that we have shown to be correct [10, Ch. 5] are listed below.

- A hard or soft basic CPU reservation with amount x and period y may be converted, for any c, into a soft, continuous reservation with amount x and period $(2y - x + c)$.
- A continuous CPU reservation may be converted into a basic CPU reservation with the same amount and period. The basic reservation will be hard if the continuous reservation was hard and will be soft otherwise.
- A hard or soft continuous CPU reservation with amount x and period y may be converted into a proportional share guarantee that has share x/y and error bound $(x/y)(y - x)$.
- A hard or soft basic CPU reservation with amount x and period y may be converted into a proportional share guarantee that has share x/y and error bound $2(x/y)(y - x)$.
- A proportional share guarantee with share s and error δ may be converted, for any $y \geq \delta/s$, into a basic or continuous soft CPU reservation with amount $(ys - \delta)$ and period y. (This conversion was first presented by Stoica et al. [15].)

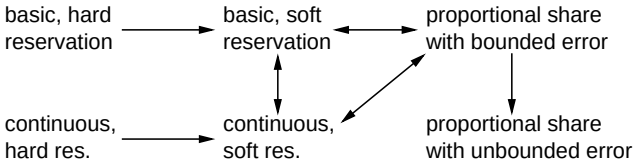

Figure 2: Permissible conversions between soft real-time scheduling guarantees

These conversions, depicted in Figure 2, give CRM the ability to provide diverse guarantee semantics given any one of basic CPU reservations, continuous CPU reservations, or proportional share scheduling with bounded error. However, when scheduling real applications, the number of conversions should be minimized. There is a genuine semantic mismatch between the different kinds of guarantees and conversions can therefore result in wasted CPU time.

4. ENFORCING USER-SPECIFIED RULES

CRM will provide users and user-specified rules with an organized collection of hooks and reflective information about the system that they can use to make decisions about the allocation of processor time. The following entities exist within our CRM design:

- **Resource principals** represent entities whose aggregate resource usage is to be controlled: for example, users, administrative domains, or accounting domains. Resource principals are lightweight: they are not hierarchical and every application in a system belongs to exactly one principal.
- **Guarantees** are held by each application thread in the system. Each guarantee is owned by a single resource principal.

- **Requests** for guarantees are made by applications. Requests have first-class status within CRM since requests that are not immediately granted may remain pending until resources become available.
- **Processor allocation rules** are either system-wide or are owned by a particular resource principal. They determine how CPU time allocated to a principal is to be sub-allocated.
- **Events** cause rules to be evaluated. Events are named. Rules may be configured to run when certain events are signaled, and may signal additional events. Some events are built-in; for example, the `new_request` rule is signaled whenever an application requests real-time service or when CRM determines that such service should be requested for the application.

Rules are used to enforce high-level policies about the allocation of CPU time; for example, to enforce fairness between users on a multi-user machine, to run the set of applications that is the most important to a user, to run the feasible set of applications that maximizes total importance, or to select an appropriate mode for an adaptive application. Still to be addressed are the issues of ensuring that rules execute in an order that makes sense, and otherwise detecting or preventing unintentional interference between rules.

5. OTHER FUNCTIONALITY

CRM includes the following additional functionality.

Gathering application timing information. Although the period of a real-time application depends only on the structure of the application, the required amount of CPU time depends on many factors: the speed and model of the CPU, the particular data being processed, and characteristics of the operating system and device drivers. For example, a game may require far less CPU time when it has the use of a powerful 3D accelerator than when the graphics subsystem must perform rendering in software. In most cases it is impossible to predict, a priori, the CPU requirements of a given application. Rather, they must be measured on a particular system.

Our planned strategy for performing this measurement is to run each newly installed application in isolation and to measure the amount of CPU time that it requires. There are a variety of techniques that can be used to do this even when no direct instrumentation has been added to the OS: e.g. timed waits (used by applications to implement periodic tasks) can be monitored using standard debugging hooks. Debugging hooks can also be used to determine the priority and starting address of each application thread. The start address can be used to distinguish between threads during later program invocations, and the priority can be interpreted as a hint about relative application importance. Finally, CRM can run an instrumented thread at low priority to determine an application's overall CPU requirements by measuring the amount of slack time in the schedule.

Once the CPU requirements for an application have been determined, its scheduling parameters can be estimated by mapping the requirements to one of the scheduling abstractions that we described in Section 3. One possible heuristic for doing this would be to assign a basic, soft CPU reservation to an application where the period is equal to the observed application period, and an amount equal to the observed amount of CPU time per period plus a small fudge factor.

Interacting with the user. Although we believe that end-users should have as little to do with the resource manager as possible, some interaction will be necessary. Such interaction could take place through very simple interfaces. For example, a pie chart could be used to describe the resource usage of the set of running applications to the user, and a mouse click on a poorly performing application could be used to notify CRM that the user wishes the application's performance to be improved.

Storing timing information and user preferences. CRM will interact with one of the many free, lightweight database packages to store information about application requirements and user preferences.

Interacting with the scheduling subsystem. We will provide a back-end (implemented as a dynamically linked library) for CRM for each scheduler that it supports. Initially, we plan to support some or all of the following QoS-enhanced versions of Linux: Linux/RK [9] and Linux-SRT [4], which both provide hard and soft basic CPU reservations; RED-Linux [18], which has a flexible scheduling subsystem and could be programmed to provide any kind of CPU reservation; and QLinux [17], which provides hierarchical proportional share scheduling with bounded error.

6. RELATED WORK

Previous middleware resource managers such as the QoS Broker [8], the Dynamic QoS Resource Manager [1], and the modular resource manager for Rialto [5] have tended to focus on adaptive applications and resource management in a distributed system. Unlike these systems, CRM is designed to be portable between OSs that provide different scheduling abstractions, and takes the viewpoint that few desktop applications are capable of automatic adaptation.

RT-CORBA 1.0 [13] assumes that systems it runs on provide fixed-priority scheduling, and it provides a consistent view of priorities across a distributed heterogeneous system. Recent RT-CORBA work [3] is more ambitious with respect to scheduler diversity and has the goal of ensuring semantic correctness of distributed real-time applications even when different nodes in a network run completely different scheduling algorithms.

Although there is overlap between RT-CORBA and CRM, the goals of the two systems are somewhat different. RT-CORBA is designed to support real-time computation on heterogeneous distributed systems, whereas CRM is designed to enable a graceful transition from traditional closed-system scheduling techniques to those that provide meaningful guarantees to applications in an open system.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have described the design of CRM: a middleware application that manages the allocation of CPU time in soft real-time operating systems. It will have the ability to convert between different soft real-time scheduling abstractions and can also enforce user-specified rules about the allocation of processor time. Furthermore, CRM will interact with the user and have the ability to determine and store application requirements, and to store information about user preferences.

Once we have implemented a basic version of CRM, there will be many avenues for future work. First, we could port it to other QoS-enabled operating systems. Second, we would like to integrate CRM with existing middleware such as the X Window System, the GNOME and KDE desktop environments, and real-time CORBA systems. Third, modules for gathering feedback from applications about performance metrics such as frame rate, missed deadlines, number of page faults, and the status of queues and buffers should be added to CRM. Fourth, CRM could be extended to support the scheduling of other resources such as memory, disk bandwidth, and network bandwidth. And finally, although we argued in [10] that the scheduler conversion rules in Section 3 are correct, we plan to develop a formal type system for describing real-time schedules and converting between them.

8. ACKNOWLEDGMENTS

The authors would like to thank Eric Eide and Alastair Reid for their helpful feedback on a draft of this paper.

9. REFERENCES

[1] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 307–317, Madrid, Spain, Dec. 1998.

[2] J. Bruno, J. Brustoloni, E. Gabber, B. Özden, and A. Silberschatz. Retrofitting quality of service into a time-sharing operating system. In *Proc. of the USENIX Annual Technical Conf.*, Monterey, California, June 1999.

[3] A. Corsaro, D. C. Schmidt, C. Gill, and R. Cytron. Formalizing meta-programming techniques to reconcile heterogeneous scheduling policies in open distributed real-time systems. In *Proc. of the 3rd International Symposium on Distributed Objects and Applications*, Rome, Italy, Sept. 2001.

[4] D. Ingram. *Integrated Quality of Service Management*. PhD thesis, University of Cambridge, Aug. 2000.

[5] M. B. Jones, P. J. Leach, R. P. Draves, and J. S. Barrera, III. Modular real-time resource management in the Rialto operating system. In *Proc. of the 5th Workshop on Hot Topics in Operating Systems*, May 1995.

[6] M. B. Jones and J. Regehr. CPU Reservations and Time Constraints: Implementation experience on Windows NT. In *Proc. of the 3rd USENIX Windows NT Symposium*, pages 93–102, Seattle, WA, July 1999.

[7] M. B. Jones, D. Roşu, and M.-C. Roşu. CPU Reservations and Time Constraints: Efficient, predictable scheduling of independent activities. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, pages 198–211, Saint-Malô, France, Oct. 1997.

[8] K. Nahrstedt and J. Smith. The QoS broker. *IEEE MultiMedia*, 2(1):53–67, Spring 1995.

[9] S. Oikawa and R. Rajkumar. Portable RK: A portable resource kernel for guaranteed and enforced timing behavior. In *Proc. of the 5th IEEE Real-Time Technology and Applications Symposium*, pages 111–120, Vancouver, Canada, June 1999.

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

[11] J. Regehr, M. B. Jones, and J. A. Stankovic. Operating system support for multimedia: The programming model matters. Technical Report MSR-TR-2000-89, Microsoft Research, Sept. 2000.

[12] J. Regehr and J. A. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proc. of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, London, UK, Dec. 2001.

[13] D. C. Schmidt and F. Kuhns. An overview of the real-time CORBA specification. *IEEE Computer*, 33(6), June 2000.

[14] B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus. A firm real-time system implementation using commercial off-the-shelf hardware and free software. In *Proc. of the 4th IEEE Real-Time Technology and Applications Symposium*, Denver, CO, June 1998.

[15] I. Stoica, H. Abdel-Wahab, and K. 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, Feb. 1997.

[16] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. 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, Dec. 1996.

[17] V. Sundaram, A. Chandra, P. Goyal, P. Shenoy, J. Sahni, and H. Vin. Application performance in the QLinux multimedia operating system. In *Proc. of the 8th ACM Conf. on Multimedia*, Nov. 2000.

[18] Y.-C. Wang and K.-J. Lin. Implementing a general real-time scheduling framework in the RED-Linux real-time kernel. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, pages 246–255, Phoenix, AZ, Dec. 1999.