Resource Management for a Virtual Planning Room

Gary Nutt, Toby Berk[*], Scott Brandt,
Marty Humphrey[†], and Sam Siewert

Department of Computer Science, CB 430
University of Colorado
Boulder, CO 80309-0430
nutt@cs.colorado.edu

May, 1997

Abstract

In the last few years we have been prototyping a multiperson virtual environment using low-cost workstations interconnected with relatively high-speed networks. This domain makes use of interactive and on-demand continuous media in addition to a number of other tasks that fall on a spectrum between hard real-time and best-effort response. A brute force implementation of applications in this type of domain demands excessive system resources, even though the actual requirements by different parts of the application vary according to the way the virtual environment is being used at the moment. A more sophisticated approach would be to provide both the user and the applications with the ability to dynamically adjust resource requirements according to their current needs and the availability of system resources. This position paper summarizes the nature of the support required from the operating system, then describes our approaches to resource management based on these requirements.

[*]Permanent address is School of Computer Science, Florida International University, Miami, FL
[†]Department of Computer Science and Engineering, University of Colorado at Denver, Denver, CO

1 The Application Domain

To support collaboration among human users, computers need to provide an effective set of domain-specific tools embedded in a free-form communication environment [18]. The *virtual planning room* (VPR) [19] is a multiperson virtual environment (VE) supporting free-form communication in a manner similar to electronic meeting rooms [7, 23] and other virtual environments [1, 3, 5]. It differs from most other virtual environments in that it is designed to be extended with "domain-specific tools." For example, in [17] we describe how a formal workflow/process modeling system can be embedded in the VPR to focus on group coordination. We are also currently building agent-based software to extend the VPR so that a group can navigate and control a blimp with its own onboard computer, navigation equipment, and videocamera ("FLOATERS") [21].

1.1 The Virtual Planning Room

A VPR world is defined by a collection of objects, with visual representations and behaviors of varying complexity. Domain-specific tools are added to the VPR by incorporating additional objects having "complex" behavior. The fundamental role of the VPR is to to provide real-time audio and video support across the network, to render objects on each user's screen, and to provide an environment in which to add domain-specific extensions.

The VPR is a client-server system where each person uses a client workstation to implement the human-computer interface. Hence, the client must render each visible artifact from a (VRML) representation in the corresponding object, and cause behaviors (such as modifications to objects) to be reflected in all other appropriate clients.[1]

For our prototype, we have used widely-available system software and interfaces: CORBA for the object interface, UNIX for the system call interface, and various network interfaces to support streams. This has allowed us to explore the VPR design, implementation, and functionality, even though the performance of the prototypes is severely limited by the hardware resources. Despite the number of papers focusing on VE/VR functionality, design, and user interfaces, (e.g., see [2, 11, 12]) there is surprisingly little on the effect the operating system has on the VE's performance. Now that we have a rudimentary VPR, our goal is to explore system software design and organization are well-suited for this rapidly-evovling application domain.

1.2 Issues

In developing the VPR, we have encountered a number of barriers to providing efficient application of the hardware to the VPR and its applications. For each item, current OSs fail to adequately deliver the support we need.

[1] This design implies the existence of a facility for distributed objects, described in [19], but not considered further in this position paper.

High-performance graphic rendering of diverse types of artifacts. A client machine is normally expected to render 30 frames/second, independent of the nature of the VRML descriptions and of other load on the machine. If the processing load is too large, frames will be lost. If the VPR knows frames will be dropped, it could simplify some of the images; i.e., the VPR should be able to tradeoff the quality of the rendering of some objects to preserve the frame rate, or to decide to drop the frame rate and preserve a minimum quality of the image for certain objects. The VPR needs an indication from the OS of its ability to service the load.

Supporting continuous media. Audio and video streams flow among VPR objects. While protocols can make assurances regarding the isochronous network transfer rate, only a few OSs attempt to make guarantees regarding the throughput rate through the OS itself [4, 8, 9, 14, 15, 16]. We desire VPR applications to be able to tradeoff loss, jitter, and latency in each stream with other activity in the VPR.

VPR applications. While it might be technically possible to statically tune an operating system to provide optimized support for the appearance and behavior of each object in a particular VPR, when extensions are added, the tradeoffs change according to the requirements of these extensions as well as the VPR. A better approach would allow the VPR and its extensions to influence the tradeoffs on resource allocation that must be made by the OS.

Resource Allocation Philosophies. The VPR and similar multimedia applications produce transient loads on the system's resources, often resulting in overload conditions. We desire an OS that takes on certain characteristics of embedded software — the application assumes part of the responsibility for the resource allocation strategy — yet which fits within the general framework of a multiprogrammed OS. We recognize that the brute force approach is for each component of the VPR to acquire (and perhaps even use) the maximum amount of resources it will ever need at all times. However, this approach leads to excessive hardware requirements that can only be met through over-allocation. Instead, we advocate a programming environment in which sophisticated applications dynamically adjust their requirements according to the availability of resources and the activity directed by the user. If a part of the VPR is dormant, then we do not want to expend much resource on supporting it; if it is active, we want to direct as much resource to the component as necessary.

We base our approach on quality of service (QoS) contracts between application components and the OS. A contract can change according to changes in the overall system load — the QoS is *dynamic*. When the situation changes, the application repertoire and the OS jointly choose a new QoS contract through *negotiation*. We desire resource allocation policies based on a *dynamically negotiated QoS*.

Others have also recognized that this kind of shift in the application-OS interface could substantially improve overall system performance for "single-application, multiprogrammed" domains such as the VPR e.g., see [20]. In the remainder of this paper, we describe some of the approaches we are exploring for providing a dynamically negotiated QoS interface.

2 In-Kernel Pipes

Continuous media support at the operating system level has focused on ways to provide application code with access and control of kernel-level data, e.g, see [4, 6, 8]. While we believe these mechanisms are necessary, they do not appear to be sufficient to ensure management of the deadline-sensitive aspects of continuous media processing; real-time techniques need to replace best-effort techniques in the mechanism. We are building a real-time, parametrically-controlled in-kernel pipe (RT-PCIP) mechanism, used in conjunction with an execution-performance agent (EPA) tool, to manage tasks that can naturally be implemented in a pipeline architecture [22].

Our RT-PCIP allows pipe modules to be dynamically inserted into a device-device stream, operating under the control of the kernel-level EPA. The EPA occasionally gets parameters from user-space applications, then computes a set of directives for how the modules should be processed vis-a-vis their real-time requirements.

The RT-PCIP design is most similar to Fall's in-kernel pipe design [6], the extensions being in the real-time management of the pipe stage execution. In continuous media applications, information arrives on one device (usually the network), and must be written to another device under relatively stringent isochronous deadlines. Often, only a small amount of processing needs to be done to the incoming information before it can be written to the outgoing device. One approach is to map the kernel device buffers into the address space of user level threads (e.g., [4, 8]) then allow the user threads to filter the data prior to writing it to the source device. With in-kernel pipes, a set of modules is inserted into kernel space to be executed by a corresponding set of kernel threads.

In a general-purpose OS environment, pipe module execution is controlled by a kernel thread scheduler — typically a best-effort scheduler. As long as the system does not become overloaded, the pipe facility will provide satisfactory service. To address overload conditions (a situation we commonly observe in the VPR), the EPA provides dynamically-computed priorities to the scheduler so that it can allocate the CPU to threads with imminent deadlines.

Hard real-time system technology has been developed in domains where the OS must *guarantee* that each task admitted to the system can be completed prior to a prespecified *deadline*. Such systems are, of necessity, conservative: Task processing estimates are expressed in terms of the worst case execution time (WCET), admission is based on the assumption that every task uses its maximum amount of resources, and the schedule ensures that all admitted tasks execute by their deadline.

Continuous media applications have less stringent deadline requirements: The threads in a continuous media pipe must *usually* meet deadlines, but it is acceptable to occasionally miss one. In the our approach, when the system is overloaded — the frequency of missed deadlines is too high — the EPA reduces the loading conditions by reconfiguring the pipeline, e.g., by removing a compression filter (trading off network bandwidth for CPU bandwidth).

The EPA design is driven by experience and practicality: Rather than using WCET for computing the schedule, we use a range of values with an associated confidence level to specify the execution time. The additional requirement on the "application" is to provide

execution time estimates with a range and a confidence; this is only a slightly more complex approach than is used in Rialto [14].

In [22] we elaborate on the RT-PCIP and EPA designs, including a more complete explanation of the EPA operation. For this position paper, we provide only the overview. This aspect of our approach is a special case of our related work on scheduling and resource management.

3 Soft Real Time Processing

Our need for supporting objects operating under deadlines extends beyond in-kernel pipe support, since the VPR must support multimedia in other contexts. For example as mentioned earlier, the OS should allow the VPR/user to manipulate the VRML display list (reducing the number or complexity of the presentation of objects) to reduce the load on the graphic processor.

We have been influenced by studies in which user-space software provides information beyond priority, WCET, and deadline as part of its resource request. Jensen et al. proposed that an application provide a *value function* to assist the scheduler to meet soft deadlines [13]. Mercer et al. carefully distinguish between admission and scheduling of resources with the *reserve* function in RT-Mach, essentially making harder guarantees at admission time than at scheduling time [15]. Rialto extends the RT-Mach approach by allowing the programmer to query the OS regarding its ability to meet a deadline for each phase of work [14].

Our approach is based on soft deadlines and QoS. The goal is to allow a QoS assurance to be negotiated between an application (object) and the OS based on the application's need for service and the utilization level of the resource in the system.[2] The assumption is that if one or more applications begin to miss their deadlines, then the system has become overloaded so the individual QoS levels for all applications should be adjusted.

Our multimedia applications are written with multi-level functionality where the QoS assurance level is related to the functionality level; if the application can only be assured of low resource availability then it should do less work. In the context of graphics, this might mean that high CPU assurance allows the application to do high fidelity presentation; but low CPU assurance forces an application to render graphics with low fidelity presentation, e.g., wire frame. An application requests service by providing its deadline and desired functionality level; the QoS manager negotiates the functionality level with the application (or its agent), arriving at a level at which the application will run. Because of the adaptive nature of the QoS assurance, we do not require an execution time estimate, though clearly that would simplify the dynamic behavior of the assurances. The application chooses its execution strategy as a function of the negotiated level. When any application misses a deadline, the QoS manager reevaluates the negotiated levels (using a specific policy). Similarly, if the system does not result in any missed deadlines over a period of time and some applications are executing at lower than their maximum functionality level, then the manager can nego-

[2] Our work focuses on the CPU for now, though our intent is to extend this to other shared resources.

tiate levels upwards (using a specific policy). See [10] for more information about this aspect of the project.

A more comprehensive approach is for the application to provide even more information about its behavior, then to allow resource allocation to take advantage of this information. In this approach, the application provides a *benefit curve* (similar to the Alpha kernel value curve [13]) relating the value of a task's result with the time that the result is delivered. If a result is delivered within a deadline, it has a maximum benefit to the application; as time passes, the value decreases, potentially eventually causing damage. In a hard real-time application, the benefit curve is a step function with soft and hard deadlines being the same; normally, the application is terminated if the response time exceeds the deadline. Non real-time interactive processes can also be represented in this framework with increasing benefit curves representing increasing urgency of a process since the time that it was last scheduled.

We are exploring an approach whereby we use each benefit curve directly as part of the scheduling mechanism, i.e. an application provides it's own benefit curve as part of it's negotiation for CPU service from the OS. The scheduler meets all deadlines if possible; otherwise, the relative importance of the applications, the shape of their benefit curves, and the details of the timing determine the order in which tasks get executed and consequently which tasks' deadlines will not be met and by how much.

The benefit curve must be specified by the application at the time that a process enters the system, but it can also change as the process runs, i.e. the curves may be changed depending on the status of the applications.

4 Status

The VPR has been under development for about two years, with the focus being on the functionality of the system. It can be used to demonstrate multiperson interactions in a common virtual environment, though the performance is severely limited by the factors we explain in this paper. We have learned a considerable amount about the demands this kind of application makes on its underlying support system. We are now moving into a phase where we must focus on operating system tuning and design.

Recently we began defining our requirements for the OS, and considering how it can be designed to meet these requirements. This work is still in the formative stages at this point with only preliminary results, though it is now sufficiently advanced to report to the community for feedback and ideas.

References

[1] Denis Amselem. A window on shared virtual environments. *Presence: Teleoperators and Virtual Environments*, 4(2):130–145, 1995.

[2] Special Issue of AT&T Technical Journal on Multimedia, September/October 1995. Nikil Jayant, Technical Reviewing Editor.

[3] David A. Berkley and J. Robert Ensor. Multimedia research platforms. *AT&T Technical Journal*, 74(5):34–45, September/October 1995.

[4] Geoff Coulson, Andrew Campbell, Philippe Robin, Gordon Blair, Michae Papathomas, and David Hutchinson. The design of a QoS controlled ATM based communication system. *IEEE JSAC Special Issue on ATM Local Area Networks*, 1994.

[5] Lennart E. Fahlen, Charles Grant Brown, Olov Stahl, and Christer Carlsson. A space based model for user interaction in shared synthetic environments. In *Proceedings of Interchi '93*, pages 43–48, April 1993.

[6] Kevin Fall and Joseph Pasquale. Exploiting in-kernel data paths to improve i/o throughput and cpu availability. In *Proceedings of the Winter 1993 USENIX Conference*, pages 327–333, January 1993.

[7] Jania Gajewska, Jay Kistler, Mark S. Manasse, and David D. Redell. Argo: A system for distributed collaboration. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 433–440, 1994.

[8] Ramesh Govindan and David P. Anderson. Scheduling and IPC mechanisms for continuous media. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, pages 68–80, 1991.

[9] Pawan Goyal, Xingan Guo, and Harrick M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI'96)*, pages 107–121, 1996.

[10] Marty Humphrey, Toby Berk, Scott Brandt, and Gary Nutt. Dynamic quality of service resource management for multimedia applications on general purpose operating systems. Technical report, Department of Computer Science, University of Colorado, June 1997. submitted for publication.

[11] Special Issue of IEEE Computer on Virtual Environments, July 1995. David R. Pratt, Michael Zyda, and Kristen Kelleher.

[12] Special Issue of IEEE Computer on Multimedia Systems and Applications, May 1995. Arturo A. Rodriguez and Lawrence A. Rowe, Guest Editors.

[13] E. Douglas Jensen, C. Douglass Locke, and Hideyuki Toduda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 112–122. IEEE, 1985.

[14] M. B. Jones, P. Leach, R. Draves, and J. Barbera III. Support for user-centric modular real-time resource management in the Rialto operating system. In *Proceedings of the NOSSDAV'95*, 1995.

[15] Cliff Mercer, Stephan Savage, and Hideyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 90–99, May 1994.

[16] Jason Nieh and Monica S. Lam. The design of SMART: A scheduler for multimedia applications. Technical Report CSL-TR-92-697, Computer Systems Laboratory, Stanford University, June 1996.

[17] Gary J. Nutt. Model-based virtual environments for collaboration. Technical Report CU-CS-799-95, Department of Computer Science, University of Colorado, Boulder, December 1995.

[18] Gary J. Nutt. The evolution toward flexible workflow systems. *Distributed Systems Engineering*, 3:276–294, 1996.

[19] Gary J. Nutt, Joe Antell, Scott Brandt, Chris Gantz, Adam Griff, and Jim Mankovich. Software support for a virtual planning room. Technical Report CU-CS-800-95, Department of Computer Science, University of Colorado, Boulder, December 1995.

[20] Shuichi Oikawa and Ragunathan Rajkumar. A resource centric approach to multimedia operating systems. In *Proceedings of IEEE Real-Time Systems Symposium Workshop on Resource Allocation Problems in Multimedia Systems*. IEEE, December 1996.

[21] Sam Siewert. Operating systems support for parametric control of isochronous and sporadic execution streams in multiple time frames, 1996. Ph.D. dissertation proposal.

[22] Sam Siewert, Gary J. Nutt, and Marty Humphrey. Real-time parametrically controlled in-kernel pipelines. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium*, June 1997. to appear.

[23] Harrick M. Vin, Polle T. Zellweger, Daniel C. Swinehart, and P. Venkat Rangan. Multimedia conferencing in the etherphone environment. *IEEE Computer*, 24(10):237—268, October 1991.