A Discrete and Dynamic Approach to Application/Operating System QoS Resource Management

Scott Brandt, Gary Nutt, and Ken Klingenstein
University of Colorado at Boulder

1. Introduction

One of the primary goals of Internet2 is to enable widespread use of high-bandwidth and time sensitive distributed applications. Such applications as desktop video-over-IP and distributed virtual cubicles will require guarantees of consistent performance from the lower layers of the supporting computing environment. Service committments are needed from both of the major components, namely operating systems and networks. At the network layer, important service requirements appear to be bandwidth, delay and jitter. At the operating system layer, service requirements may include cpu, memory and buffer commitments. For real-time applications to be successful, they will need to negotiate levels of service with both networks and operating systems, and respond to changes in the service levels actually provided.

In networking, quality of service issues appear to center around methods of handling congestion, i.e. allocating resources at different service levels to different flows. For operating systems, an analogue appears true as well, that quality of service depends primarily on schemes for allocating system resources to different applications. In a modern computing environment, multiple concurrent applications may be competing for operating resources, notably cpu, and the provision of such resources to applications, and the adaptation of the applications to changes in those resource levels, will be critical tothe viability of real-time applications.

Operating systems designers have been creating mechanisms to support QoS-based soft real-time application execution. These mechanisms provide a variety of interfaces for determining the amount of resources that will be allocated to an application, allowing a process to a) *negotiate* with the operating system for a specific amount of resources as in RT Mach [7] and Rialto [5][6]; b) specify a *range* of resource allocations as in MMOSS [3]; or c) specify a measure of application *importance* that can be used to compute a fair resource allocation as in SMART [8][9]. These systems all provide a mechanism that can be used to dynamically reduce the resource allotment granted to the running applications. In the extreme, the applications may be forced to dynamically adapt to a strategy in which the resource allocation is less than that required for average-case execution. In creating these resource management mechanisms, operating systems developers have assumed that it is possible for applications to adjust their behavior according to the availability of resources, but without providing a general model of application development for such an environment.

We are exploring a middleware solution which takes an approach in which applications cooperate with the operating system in their use of system resources, adapting to the current state of the system to maximize the benefit obtained from the available resources. This approach is in contrast with the operating system approach in which utilization outside the worst case requires enforcement. This distinction -- negotiation and adaptation versus strict enforcement -- is a major philosophical difference in our approach when compared to existing approaches to achieving quality of service from an operating system.

In previous papers [1][4][10][11] we presented *execution levels*, a method for dynamically managing soft real-time application execution in an environments with varying QoS allocations. We have demonstrated the feasibility of cooperative middleware based QoS management and discussed our prototype middleware execution level based QoS resource manager called the *Dynamic QoS Resource Manager* (DQM) and examined a set of representative QoS allocation algorithms within this context.

In continuing our research in this area, we have extended the DQM in several ways that bring it significantly closer to our goal of having a viable middleware QoS resource management agent [2]. In our earlier work we used synthetic programs to experiment with the DQM, though now our tests are driven using working applications (two MPEG players). We have also expanded the adaptive capabilities of the DQM with a technique called *dynamic estimate refinement*.

It is important to note that while these techniques directly address only the QoS for application to operating system interactions, it appears that the approach can be extended to application to network QoS interactions. One possible extension would be for applications to utilize similar tools to negotiate levels for QoS with network layer devices, and, in turn, receive updates from the net for application adaptation. An alternative approach would be to include network layer interactions as part of the operating system interaction, and consider network capacity as one of the several resources that the operating system and application negotiate. It is widely thought that operating system QoS, rather than network QoS, may be the greatest factor in end-to-end performance; if so, then the latter approach has much appeal.

2. Execution Levels and the DQM

Our research has focused on supporting soft real-time processes on general-purpose operating systems. Most soft real-time systems soften the real-time behavior of the applications by moderating the percentage of missed deadlines or the amount by which deadlines are missed, with smaller amounts considered better. This is adequate for the class of soft real-time processes for which missed deadlines are acceptable, but not all such processes fall into this category. For example, desktop playback of a fixed-bandwidth network-based continuous media stream does not allow for all deadlines to be missed by a certain amount because eventually the OS will run out of buffer space to hold the queue of frames that is slowly backing up. In this case a preferable solution would be one such as dropping frames or reducing the amount of processing for each frame so that the hard deadlines (enforced by the arrival of new data) can still be met.

In support of application-controlled adaptive behaviour, we have developed the execution level based application execution model. With execution levels, each application is constructed using a set of algorithms for achieving its goals, ordered by their relative resource usage and the relative quality of their output. The execution levels are the application specification of the soft real-time policy it implements. The QoS manager provides the mechanism for managing the soft real-time execution of the applications by managing the dynamically adjusting the level of each running application. Avariety of QoS allocation algorithms exist that adjust the applications according to the currently available resources.

In order to examine QoS-based soft real-time processing with the execution level model, we have developed a prototype system consisting of a middleware DQM and a library of DQM interface and soft real-time support functions called the Soft Real-Time Resource Library (SRL). This pro-

totype system has allowed us to experiment with execution level based adaptive soft real-time processing. Like the flexible QoS systems cited above, the current implementation of our DQM works solely with the CPU resource. However, we believe that the concepts described in this paper can be extended in a straightforward manner to encompass other resources such as network bandwidth and memory.

The DQM dynamically determines a level for the running applications based on the available resources and the specified benefit of the application, and changes the level of each running application until all applications run without missing deadlines, the system utilization is above some predetermined minimum, and stability has been reached. Resource availability (or the lack thereof) is determined in a few different ways. CPU overload is determined by the incidence of deadline misses in the running applications. The SRL linked into each application notifies the DQM each time an application misses a deadline. CPU underutilization is determined by examining CPU idle time. System idle time can be determined in several ways including via the operating system, through the /proc file system, by measuring the CPU usage of a low priority application, and by taking the complement of the CPU usage measurements (or estimates) of the running applications. If the operating system provides idle time information, this information is the most reliable.

3. Results

In order to examine the operation of the DQM we have used both synthetic and real applications. The synthetic applications consume resource according to their level specifications without performing any useful work. They have allowed us to exercise the DQM with a wide variety of level specifications.

The two real applications are mpeg players, but they differ in the way that their real-time behavior has been softened. The first application changes the frame rate of the displayed image from 0 frames/second to 10 frames/second. This particular application required no algorithmic changes other than the inclusion of the three SRL functions: dqm_init(), called once at the beginning of the application; dqm_loop(), called each time through the main loop; and dqm_exit(), called at application exit. The second application dynamically adjusts the size of the image displayed on the screen. Since the amount of work is related to how much time is spent drawing the pixels on the screen, this results in a reasonable range of CPU usage numbers over the different levels. Dqm_loop() returns the level at which the application should execute, so the main control loop of the application contains a switch which sets the size of the displayed image accordingly.

As shown elsewhere [1][2], our results to date have been very promising. The applications adapt quickly to changing resource availability and stabilize at appropriate execution levels as determined by the DQM.

4. Conclusion

Internet2 based applications will make tremendous demands for the use of the host computer and network resources. Information must flow from an application through the host operating system, over the internet, up through a second host operating system, and into a receiver application. Because of the heavy demand for various resources to support this data movement, these applications must be prepared to participate in the way the resources are managed across the applica-

tion's components (as well as across applications). The execution level model and DQM demonstrate a sound approach for managing the resources in a soft real-time environment within a host machine.

The contribution of this work to the Internet2 community is clear. Execution levels are a simple and natural model for developing resource-sensitive adaptive applications and the DQM demonstrates one way in which a host can support the execution of such applications. We have illustrated the utility of this new approach to resource management -- an approach that can be applied to network bandwidth as well as CPU resources. Just as host machines are oversubscribed for their resources, the internet is (and will continue to be) oversubscribed for its bandwidth. In such an operating environment, host computers must be able to administer the network bandwidth so as to provide assurances for service rates without making hard QoS guarantees. We believe execution levels and the DQM can be the basis of a sound approach for managing the allocation of Internet2 bandwidth.

5. Acknowledgments

Scott Brandt and Gary Nutt were partially supported by NSF grant number IRI-9307619.

6. References

[1] S. Brandt, G. Nutt, T. Berk, and M. Humphrey. *Soft Real-Time Application Execution with Dynamic Quality of Service Assurance*. Proceedings of the Sixth IEEE/IFIP International Workshop on Quality of Service, pp. 154-163, May 1998.

[2] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. *A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage*. Submitted for publication, May 1998.

[3] C. Fan. *Realizing a Soft Real-Time Framework for Supporting Distributed Multimedia Applications*. Proceedings of the 5th IEEE Workshop on the Future Trends of Distributed Computing Systems, pp. 128-134, August 1995.

[4] M. Humphrey, T. Berk, S. Brandt, G. Nutt. *The DQM Architecture: Middleware for Application-centered QoS Resource Management*. IEEE Workshop on Middleware for Distributed Real-Time Systems and Services, December 1997, pp. 97-104.

[5] M. Jones, J. Barbera III, and A. Forin. *An Overview of the Rialto Real-Time Architecture*. Proceedings of the Seventh ACM SIGOPS European Workshop, pp. 249-256, September 1996.

[6] M. Jones, D. Rosu, M. Rosu. *CPU Reservations & Time Constraints: Efficient Predictable Scheduling of Independent Activities*. Proceedings of the 16th ACM Symposium on Operating Systems Principles, October 1997.

[7] C. Mercer, S. Savage and H. Tokuda. *Processor Capacity Reserves: Operating System Support for Multimedia Applications*. Proceedings of the International Conference on Multimedia Computing and Systems, pp. 90-99, May 1994.

[8] J. Nieh and M. Lam. *The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications.*Proceedings of the 16th ACM Symposium on Operating Systems Principles, October 1997.

[9] J. Nieh and M. Lam. *Integrated Processor Scheduling for Multimedia*. Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, April 1995.

[10] G. Nutt, T. Berk, S. Brandt, M. Humphrey, and S. Siewert. *Resource Management of a Virtual Planning Room*. Proceedings of the Third International Workshop on Multimedia Information Systems, September 1997.

[11] G. Nutt, S. Brandt, A. Griff, S. Siewert, M. Humphrey, and T. Berk. *Dynamically Negotiated Resource Management for Data Intensive Application Suites*. Transactions on Knowledge and Data Engineering, to appear.