Using User Hints to Guide Resource Management for Quality of Service

Michael J. Katchabaw, Hanan L. Lutfiyya, and Michael A. Bauer
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada

Abstract

Typical application usage patterns in today's computing environments often result in a significant waste of computing resources—applications that are the focus of user activity or attention can starve for scarce resources while applications that are essentially being ignored still consume them. This, in turn, unnecessarily impairs the quality of service delivered by the affected applications.

Fortunately, users often provide hints of their interest and activity through interactions with their computing environments or applications directly. By being receptive to these user hints, quality of service solutions can conserve previously wasted resources and reallocate them to applications in need, thereby improving the overall quality delivered. In this paper, we describe a quality of service solution developed to this end. Experimental results derived from the use of our prototype system are also presented.

Keywords: User hints, Application-driven quality of service, Resource management

1 Introduction

To meet increasing user needs for performance, reliability, and availability in mission critical applications, a high level of quality of service must be delivered. Given an infinite supply of all computing resources (including CPU cycles, memory, disk bandwidth, network bandwidth, and so on), we would be able to satisfy the quality expectations of all users on all applications. Since this will never be the case, providing users with expected quality of service entails effective management of the limited supply of resources that are actually available.

A study of windowing environment usage by faculty, staff, and graduate students in our department [11] revealed that users frequently ignore executing applications for extended periods of time, which can result in significant wastage of these scarce computing resources. Each user had on average 17 application windows open at any given time; less than 15% of the study group had less than 10 windows open at any given time. Less than 15% of the study group logged off when they were done a session; the rest used a screen saver or locker of some kind. As a result, these users had their screens saved or locked nearly 65% of the time, with applications still executing and consuming resources underneath. With a screen saver or lock not active, only 22% of the applications executing on average at any time were completely visible and usable to the user. Nearly 55% of the applications running were minimized, and the rest were partially or completely obscured from view, being covered by other windows. In the end, the average application spends only about 40% of its life completely visible and usable; the rest is spent either minimized or covered in some way. In this 60% of its life, however, the application still consumes resources, even though the application is essentially useless and is not providing meaningful service to users. Other studies and evaluations [2, 5, 7] echo these results.

Applications that are the focus of user interest and activity should be favoured in resource allocation and scheduling to provide the best possible quality, as opposed to those applications which users do not care about and essentially ignore. Fortunately, users often provide hints of their interest and activity through interactions with their windowing en-

Appeared in the Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, Nevada, June 1999

vironments or applications directly. For example, as demonstrated above, applications that users are not attending to tend to get minimized or covered over on the window desktop in favour of those on which the user is focussing. Unfortunately, essentially all existing quality of service solutions are not receptive to these user hints—with all applications having the same rights to quality and resources regardless of whether or not the user is giving any attention to them. Furthermore, since many existing quality of service solutions allocate resources statically throughout the lifetime of an application [3], there are no opportunities to adjust resource allocations in response to changes in user needs and activity. These problems result in resources being wasted and spread thin over some applications unnecessarily, and quality expectations for other applications not being satisfied.

To address these problems, we have developed a quality of service management system capable of detecting and making use of these user hints to guide resource management. By reallocating and rescheduling resources to applications that are the focus of user activity, quality of service can be provided where it is truly needed, thereby making more effective use of scarce resources and reducing wastage. This paper briefly describes our approach and initial experimental results obtained through a prototype system.

2 Related Work

To date, there has been relatively little research and industrial effort investigating how to effectively make use of hints of user activity to guide resource management. While these issues were raised in [1], no design or implementation work was ever completed. [14] also examines these issues, but only in the context of their own 3D landscape environment, and not generally available production environments. Research into multimedia applications such as [12, 13, 17, 18] and others has yielded adaptive applications capable of responding to user feedback (for example, by not transmitting audio streams when audio is muted by the user). Unfortunately, much of this work is domain-specific and limited in scope, quite often only to reducing work done in response to user feedback, without deallocating and descheduling resources, which can lead to further problems. From industrial research, Microsoft Windows 95, 98, and NT [16] can capture some user hints to adjust application priorities and time quanta slightly as applications are foregrounded and backgrounded. Some flavours of Unix have limited support for user hints, by adjusting scheduling according to I/O activity and so on, but this tends to be inadequate for responding to user hints in general. Consequently, there is much still to be done to address these issues.

3 Our Approach to Quality of Service Management

Figure 1 shows the key components to our system, derived from our earlier work in dynamic application-driven quality of service management [9, 10].

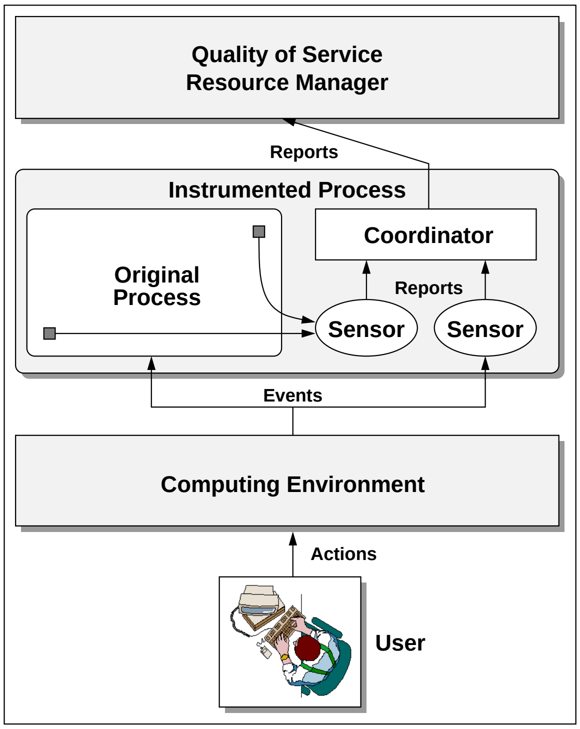

Figure 1: User Hints in Application-Driven Quality of Service Management

In our solution, quality of service expectations are captured as policies—specifications

Appeared in the Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, Nevada, June 1999

giving constraints of application behaviour in terms of a variety of metrics (for example, the desired number of frames of media delivered to the user per second). Our prototype uses object-oriented instrumentation [8] to measure metrics indicative of application behaviour and compare the measurements to policy expectations. These measurements are taken by ***sensors*** through ***probes*** inserted into the original process code. Thresholds in the sensors are used to capture policy expectations and filter between acceptable and deviant process behaviour.

If a sensor determines that policy expectations are not satisfied, the instrumentation ***coordinator*** is notified. This component collects, correlates, and filters reports and then forwards them to a ***quality of service resource manager***, or simply a resource manager, which is responsible for the allocation and scheduling of one or more resources across one or more systems, perhaps through interacting with other resource managers on remote systems. If the delivered quality of service exceeds expectations by a large margin, the application could do with a less generous resource allocation, and the resource manager makes the appropriate adjustments. On the other hand, if the delivered quality of service is below expectations, the application requires a better resource allocation from the resource manager, either through the allocation of additional resources, or a better distribution of previously allocated resources. As a result, the system is very dynamic and receptive to changing user needs and environment conditions.

Instrumentation sensors are also used to detect hints of user activity, including the minimizing and restoring of windows, covering and uncovering of windows, and activation and deactivation of screen savers and locks. Support for application-specific user hints is also provided, through the object-oriented nature of our instrumentation. New classes of sensors can be derived for particular applications; for example, to detect when audio volume has been turned down by the user of a multimedia application, indicating that this aspect of the application is no longer important. Furthermore, existing classes of sensors may require behavioural specialization for particular applications. For example, a video playback application may require minimal or no resources when it is minimized by the user. However, a web browser may be minimized by the user while it is retrieving a document over the network—taking all of its resources away so it cannot download the document in the background would not at all reflect the user's intentions and expectations. On the other hand, if the browser is not downloading a document, its resources may be safely taken away so they are not wasted on animations or applets in which the user currently has no interest. Additional fine tuning of instrumentation behaviour can be done at run time by adjusting the thresholding and filtering of data, as well as disabling and reenabling instrumentation. This allows the collection and handling of user hints from applications to be tailored as needed as user and environment conditions change over time.

When a user interacts with an application, they do so through their ***computing environment***, which is an amalgam of hardware, operating system, middleware, and windowing environment components. User activity is translated by the environment into events sent to the instrumented application. User hints are collected by sensors directly from the event stream, or indirectly from probes embedded in the original process code. When hints are detected that indicate a user is not currently interested in a particular application, the instrumentation in the application is temporarily disabled, resulting in the suspension of the quality of service policies contained within its sensors. The resources allocated to the application are consequently freed for reallocation elsewhere to applications in need as necessary by the appropriate resource managers. Likewise, policies are reinstated, by reenabling instrumentation, and resources are reallocated when user hints indicate that the application has once again become the focus of user activity. This results in providing resources to the applications that truly need them, and reducing the consumption and wastage of resources by applications that do not.

As proof of concept, we have developed instrumentation and quality of service resource management software for the Solaris operating system, supporting the detection and handling of user hints from X Windows environments. The resource manager, taken from our earlier work [9, 10], only handles the allocation of CPU cycles, as we are focussing on the detection and handling of user hints in our current work. At the moment, we are developing support for additional resources, including memory and network bandwidth, as well as support for managing and coordinating the resources of multiple systems. Ports to Java, Windows NT, and other environments are also under way. For further details of the implementation of our prototype system, please refer to [9, 10].

4 Experimental Results and Experience

This section presents results from experimentation with our instrumentation and quality of service resource management system. The experiments discussed in this section focus on the use of the Berkeley MPEG video player [15], as a considerable portion of the literature in this area uses this as a test application. This section also provides results from experiments making use of other applications such as Netscape's Mozilla/Navigator web browser [4], Id Software's popular game DOOM [6], and several desktop utilities. These experiments demonstrate that our quality of service management system improves resource management and, consequently, quality of service provided by this variety of applications through taking advantage of indicators of user interest and activity.

4.1 Resource Allocation and Scheduling Results

We began our experimentation with the Berkeley MPEG video player, as discussed above, taking the latest available version, version 2.3, and instrumenting it. To monitor its performance, we added instrumentation to measure its video playback throughput, in frames per second. We also added instrumentation to monitor generic windowing environment user hints indicating user interest and activity. The resulting program was executed on one of our UltraSparc 1 workstations using Solaris 2.5.1 and our resource manager to assist in dynamically scheduling the application. We instructed the player to display a 352×336 8-bit video file

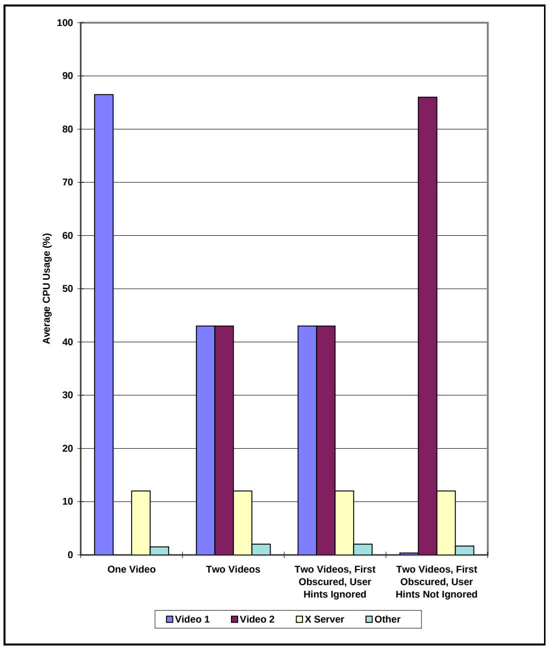

Figure 2: Video Playback CPU Analysis

from disk, and we gave it a quality of service policy dictating that it play the video with a mean frame rate of 16 frames per second, allowing for a deviation of 2 frames per second from this target (giving a range of 14 to 18 frames per second). Our experimentation consisted of executing the instrumented video player several times while ensuring the workstation was otherwise idle and isolated from other users. Experiments were replicated five times for statistical confidence.

Figure 2 shows the mean CPU usage of the UltraSparc 1 workstation in the video player experiments, measured using the `top` utility. When only one video player was executing, it consumed the bulk of CPU resources (87%) to meet its quality of service policy, with the remaining resources used by the X server and other system processes. When a duplicate video player was executed with the same workload and policy to meet, the allocation of resources to the first video player was split almost exactly in half, with the other half allocated to the new player. Obscuring (minimizing or covering) the first player had no impact on CPU usage when user hints were ignored. However, when user hints were not ignored but used by the quality of service man-

Appeared in the Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, Nevada, June 1999

agement system, obscuring the first player resulted in a reallocation of CPU resources from the first player to the second, corresponding to the user's interest in the two players. With user hints in place, only a fraction of CPU resources, approximately 0.35%, remained with the first player. This is a significant improvement, since over 43% of CPU resources are consumed unnecessarily by the first player when user hints are ignored—a tremendous waste of a valuable and scarce resource.

Further experimentation with other applications revealed similar resource wastage and possibilities for resource conservation through user hints. A single instance of Id Software's game Doom at 960x600 resolution, for instance, was found to consume approximately 89% of the CPU resources on an UltraSparc 1 workstation, whether or not the game was visible to the user. A single instance of Netscape's Mozilla/Navigator displaying a web page with the only active content being a small 80x80 pixel animated GIF image was found to consume 25% of the CPU resources on an UltraSparc 1 workstation, even when obscured[1]. On an RS/6000 220 workstation, various desktop utilities (including a clock and other tools) were found to consume approximately 15% of CPU resources, despite having the workstation locked with no one at the console. In all of these cases, when user hints were not ignored, CPU resources were reallocated to other processes in need on the system, and the CPU wastage by the above applications was minimized.

As the above experimental results have shown, taking advantage of user hints can save a substantial amount of resources and reallocate them to other processes in need. Otherwise, these resources would be wasted needlessly on applications that are not the focus of user attention and activity.

[1]As an interesting note, while this paper was being written on another UltraSparc 1 compute server in our department, the server strained under the load of another Netscape Navigator browser consuming over 87% of the CPU resources while the owner of the process was away for the weekend with his screen saver blanked out, locking his terminal. Needless to say, the authors would have been much more satisfied with their own applications if his had not been ignoring user hints!

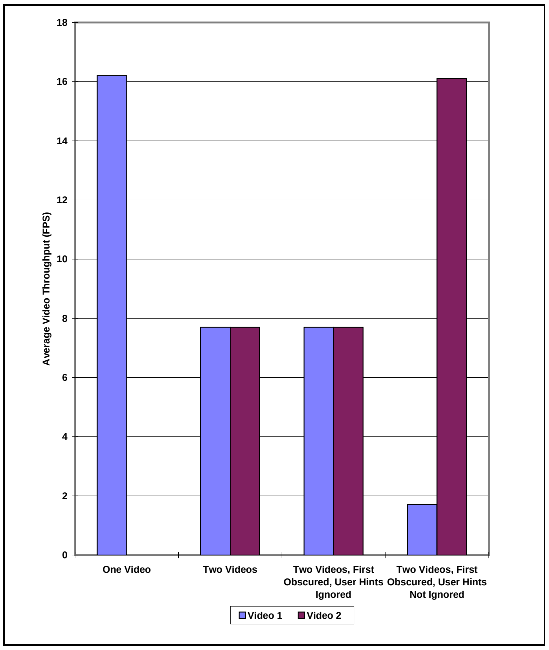

Figure 3: Video Playback Throughput Analysis

4.2 Quality of Service Provisioning Results

Having seen that user hints can lead to better resource allocation to applications in need, the experiments discussed in this section demonstrate how this, in turn, leads to improved quality of service provisioning by those applications.

Figure 3 compares the mean video playback throughput, in frames per second, for the Berkeley MPEG video player under a variety of circumstances. First, we can see that with only one player on the system, our quality of service policy was maintained by the system without any difficulty. When the second video player was introduced with the same workload and quality policy, neither could meet expectations, and the frame rate delivered was less than half of the desired level. Obscuring the first video player with user hints ignored had no impact on the quality provided, and our expectations were once again not met. However, when user hints were not ignored and were used by the quality of service management system, we can see that when the first video player was obscured, that resources allocated to the first player were freed, and reallocated to the second player. This resulted in the second player meeting its quality of service policy, with the first player performing worse than before, acceptable since it was no longer the focus of the user's attention and activity.

The above experiments were repeated using our other instrumented applications including Netscape's Navigator, Id Software's DOOM, and several desktop utilities. Experiments with these applications produced results similar to those for the Berkeley MPEG video player.

As a result, we can conclude that our quality of service management system can take advantage of user hints and reallocate resources to applications to minimize resource wastage and maximize the benefits to the user obtained from the resources. This translates into improved quality of service provisioning where it is truly needed the most by the user.

4.3 Prototype Performance Results

Measurements taken during experimentation show that the overhead cost from our approach is minimal. An instrumented process on our UltraSparc system requires approximately 400 microseconds more time to initialize itself and report to our resource manager. One pass through the instrumentation code to take a quality measurement requires only 11 microseconds on average when the instrumentation does not need to contact our resource manager. On the other hand, whenever the resource manager must be contacted, because a user hint was detected or a quality policy was violated, the instrumentation takes on average 474 microseconds to submit a report the violation to our resource manager, and to have it make appropriate resource adjustments. This is acceptable, since the alternative is even more costly in terms of wasted resources or inadequate quality provisioning.

From additional experimentation conducted with users, we found that as applications executed, quality provisioning was prompt and stable. When users gave hints of their interest and activity, resources were reallocated to the new focus of interest before any quality problems were visible to the user.

As a result, we can conclude that the prototype system carried out its task well, and at a minimal cost. For further analyses and results involving the prototype system, please consult [9, 10].

5 Concluding Remarks

The work we describe here is part of ongoing work on addressing issues in quality of service management. This paper focusses on the integration of hints of user interest and activity into the quality of service management process. We have developed an approach to detect and make use of these indicators in a dynamic quality of service management system. Using a prototype system based on this approach, we have been able to carry out a number of experiments that demonstrate that user hints can be a very successful guide to resource management. Doing so had been shown to reduce resource wastage and improve the quality of service provided by applications that are truly the focus of user attention.

We have identified several avenues for continuing and future work in this area:

- We plan to continue instrumenting applications and experimentation to further validate our work.
- We need to continue to discover additional generic hints from users that indicate their interest and activity. We also need to continue investigating new hints for specific applications and how different hints should be handled for different applications.
- It is important to continue our work to support additional resources distributed across multiple hosts, as well as our porting efforts to different platforms.

Through this work, we have gained valuable insight into the quality of service management problem. With this experience, we can further pursue solutions that effectively and completely address the variety of quality of service problems that occur in today's computing environments.

References

[1] M. Alfano. Design and Implementation of a Cooperative Multimedia Environment

with QoS Control. *Computer Communications (21), Elsevier Science*, Fall 1997.

[2] L. Bannon, A. Cypher, S. Greenspan, and M. Monty. Evaluation and Analysis of Users' Activity Organization. *Proceedings of the 1983 ACM Conference on Human Factors in Computing Systems*, Boston, Massachusetts, December 1983.

[3] H. Chu and K. Nahrstedt. A Soft Real Time Scheduling Server in UNIX Operating System. *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, Darmstadt, Germany, September 1997.

[4] Netscape Communications Corporation. Mozilla Technical Documentation. *On-line Documentation from http://www.mozilla.org*, June 1998.

[5] K. Gaylin. How are Windows Used? Some Notes on Creating an Empirically-Based Windowing Benchmark Task. *Proceedings of the 1986 ACM Conference on Human Factors in Computing Systems*, Boston, Massachusetts, April 1986.

[6] Id Software Inc. *DOOM*. December 1997.

[7] E. Kandogan and B. Shneiderman. Elastic Windows: Evaluation of Multi-Window Operations. *Proceedings of the 1997 ACM Conference on Human Factors in Computing Systems*, Atlanta, Georgia, March 1997.

[8] M. J. Katchabaw, S. L. Howard, H. L. Lutfiyya, A. D. Marshall, and M. A. Bauer. Making Distributed Applications Manageable Through Instrumentation. *Journal of Systems and Software*, In Press for 1999.

[9] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. Driving Resource Management With Application-Level Quality of Service Specifications. *Proceedings of First International Conference on Information and Computation Economies*, Charleston, South Carolina, October 1998.

[10] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. A Model for Resource Management to Support End-to-End Application-Driven Quality of Service. *Proceedings of the Tenth International Conference on Parallel and Distributed Computing Systems*, Las Vegas, Nevada, October 1998.

[11] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. A Survey of Windowing Environment Usage Patterns. *Techinical Report #540, Department of Computer Science, The University of Western Ontario*, May 1999.

[12] S. McCanne and V. Jacobson. LBNL Audio Conferencing Tool (vat). *On-line Documentation from http://www-nrg.ee.lbl.gov/vat*, December 1996.

[13] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. *Proceedings of ACM Multimedia '95*, November 1995.

[14] M. Ott, G. Michelitsch, D. Reininger, and G. Welling. An Architecture for Adaptive QoS and its Application to Multimedia Systems Design. *Computer Communications Special Issue on Guiding Quality of Service into Distributed Systems*, August 1997.

[15] K. Patel, B. Smith, and L. Rowe. Performance of a Software MPEG Video Decoder. *Proceedings of the 1993 ACM Multimedia Conference*, Anaheim, California, August 1993.

[16] J. Richter. *Advanced Windows, Third Edition*. Microsoft Press, 1997.

[17] Sun Microsystems Inc. *The Complete Guide to ShowMe 2.0.1*. Mountain View, California, May 1994.

[18] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu. A Player for Adaptive MPEG Video Streaming Over The Internet. *Proceedings of the 26th Applied Imagery Pattern Recognition Workshop*, Washington DC, October 1997.