Appears in *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*

SVR4 UNIX[‡] Scheduler Unacceptable for Multimedia Applications

Jason Nieh[†], James G. Hanko, J. Duane Northcutt, and Gerard A. Wall

[†]Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Sun Microsystems Laboratories, Inc.
2550 Garcia Avenue, MTV29-110
Mountain View, CA 94043

Applications that manipulate digital audio and video are rapidly being added to workstations. Such computations can often consume the resources of an entire machine. By incorporating a "realtime" process scheduler, UNIX System V Release 4 (SVR4), the most common basis of workstation operating systems, claims to provide system support for multimedia applications. Our quantitative measurements of real application performance demonstrate that this process scheduler is largely ineffective and can even produce system lockup. While SVR4 UNIX provides many controls for changing scheduler performance, they are virtually impossible to use successfully. Furthermore, the existence of a realtime static priority process scheduler in no way allows a user to deal with these problems. This paper provides a quantitative analysis of real system behavior, demonstrates why it is not possible to obtain the kind of behavior desired with the mechanisms currently provided by the system, and presents modifications to improve the situation.

1 Introduction

Applications that manipulate digital audio and video represent a new class of computations executed by workstation users. Audio and video applications must operate in the workstation environment without compromising the essential characteristics of the workstation. That is, audio and video applications should not reduce the workstation to a single function system, like an embedded system or single-tasking PC. Instead, the workstation operating system must manage resources in such a manner that other applications and users can continue to function correctly.

A fundamental task of any operating system is the effective management of the system's resources. Resources that must be properly managed include processor cycles, virtual and physical memory, and I/O bandwidth. Although mismanagement of any of these resources can lead to poor system function [12], we have focused on processor scheduling in this paper. Processor cycles are often the most over-subscribed resource, with many applications able to use more processing power than can be provided. In such an environment, the degree of effectiveness of processor scheduling is the dominant factor in overall system performance.

Anticipating that processor scheduling based on traditional timesharing would not be suitable for the support of multimedia applications, UNIX System V Release 4 (SVR4) provides a realtime static priority scheduler, in addition to a standard UNIX timesharing scheduler [1]. By scheduling realtime tasks at a higher priority than any other class of tasks, SVR4 UNIX allows realtime tasks to obtain processor resources when needed in order to meet their timeliness requirements. This solution claims to provide robust system support for multimedia applications by allowing applications such as those that manipulate audio and video to be placed in the real-time class. Since SVR4 UNIX is the most common basis of current workstation operating systems, it is important to investigate these assertions. Therefore, we have used an SVR4 UNIX based system to examine actual performance of real multimedia applications running in a workstation environment.

Through careful measurements of application performance, we quantitatively demonstrate that the SVR4 UNIX scheduler manages system resources poorly for both so-called realtime and timesharing activities, resulting in unacceptable system performance for multimedia applications. Not only are the application latencies much worse than desired, but pathologies occur with the scheduler such that the system no longer accepts user input. This paper describes these experiments and measurements. In addition, this paper introduce a new scheduling class which we developed that alleviates many of these problems.

The paper is organized as follows. Section 2 provides an overview of the experiments and Section 3 describes the experimental setup and applications that we used for our measurements. Section 4 presents our measurements and Section 5 discusses the results. Finally, we present conclusions and directions for future work.

2 Overview of Experiments

To examine the ability of the processor scheduling policies of SVR4 UNIX to support multimedia applications, we have identified three classes of computational activities that characterize the main types of programs executed on workstations: interactive, continuous media, and batch. Interactive activities are characteristic of applications (e.g., text editors or programs with graphical user interfaces) in which computations must be completed within a short, uniform amount of time in order not to disrupt the exchange of input and output between the user and application. Continuous media activities are characteristic of applications that manipulate sampled digital media (e.g., television or teleconferencing), via cyclic computations that must process and transport media samples at a defined rate. Batch activities are characteristic of applications (e.g., long compilations or scientific programs) in which the required processing time is sufficiently long to allow users to divert their attention to other tasks while waiting for the computation to complete. By selecting applications from each of these classes, a representative workload can be constructed that characterizes typical multimedia workstation usage. In order to simplify the experiments and the task of interpreting the resulting data, only one program from each class is used in the following experiments.

In order to obtain valid results, the experimentation was done with a standard, production workstation and operating system. However, measurements of actual system behavior are quite complex as compared to simulation-based experimentation. As a result, a number of measures were taken to permit repeatability of experiments and allow the identification and isolation of processor scheduling effects. Since the purpose of the experiments is to explore the effectiveness of various processor scheduling policies, an attempt was made to minimize the effects of other resource management decisions. Results were collected from the execution of a series of trial runs of the representative programs on the testbed hardware. The parameters of the trials were chosen so as to permit the exploration of a wide range of different conditions with the minimum number of experiments.

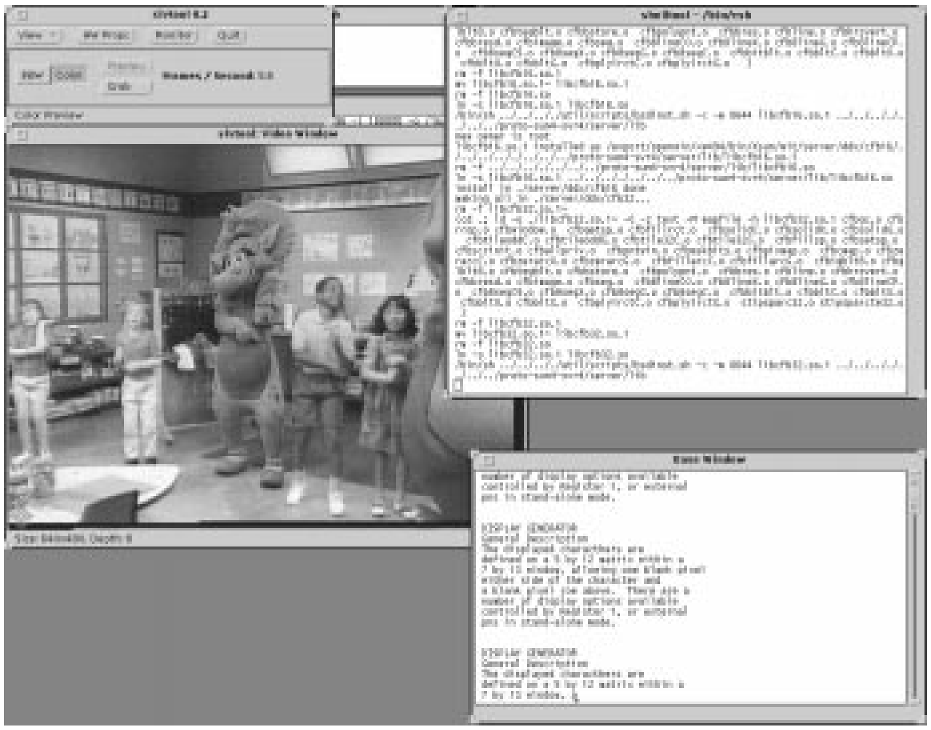
3 Experimental Design

To characterize typical workstation usage, three applications were chosen to represent interactive, continuous media, and batch activities. Each of these programs was implemented in the most obvious, and straight-forward fashion. The applications were:

- *typing* (interactive class) — This application emulates a user typing to a text editor by receiving a series of characters from a serial input line and using the X window server [10] to display them to the frame buffer.
- *video* (continuous media class) — This is a realtime video player application (e.g. as used for television, teleconferencing) that attempts to show frames of video at a constant rate. *Video* captures data from a digitizer board, dithers to 8-bit pseudocolor, and relies on the X window server to render the pixels to the frame buffer. Video frames are 640x480 pixels.
- *compute* (batch class) — This application is intended to represent programs such as the UNIX *make* utility. *make* execution is characterized by repeated spawning and waiting for various programs such as compiler passes, assemblers, and linkers. However, in order to reduce variability induced by the system's virtual memory, file system, and disk I/O handling, a simple shell script was used that repeatedly forks and waits for a small processes to complete (in this case, the UNIX *expr* command).

A number of software tools were added to the testbed to permit the logging of significant events into files, and the post-processing of these files for the generation of tracing reports. Modifications were made to the application programs and components of the system software in order to generate the necessary tracing events, but these modifications did not measurably change the performance of the software [3].

While not strictly an application program, the X window server represents a fourth major component that contributes to the overall performance of the system in these experiments. It was necessary to instrument the window server in order to obtain the desired measurements of user-level system performance. However, the window system's behavior *per se* is not of interest here, only its contribution to the user-visible performance of the application programs in the example mix.



```
[illegible]
```

```
number of display options available
controlled by [illegible] 1, or external
pin in stand-alone mode.


DISPLAY GENERATOR
General Description
The displayed characters are
defined on a 5 by 12 matrix within a
7 by 12 window, allowing one blank pixel
either side of the character and
a blank pixel line above.  There are a
number of display options available
controlled by [illegible] 1, or external
pin in stand-alone mode.


DISPLAY GENERATOR
General Description
The displayed characters are
defined on a 5 by 12 matrix within a
7 by 12 window.
```

Figure 1 Sample Application Screen

The experiments were performed in an environment representative of a typical workstation; it consisted of a SparcStation10 with a single 50MHz processor and 64MB of primary memory. The testbed system included a standard 8bit (pseudo-color) frame buffer controller (i.e., GX), and a 1GB local (SCSI) disk drive. In addition, the testbed workstation began with the current release of Sun's operating system — Solaris 2.2 [5], which is based on SVR4 UNIX.

SVR4 UNIX supports multiple concurrent scheduling policies, called *scheduling classes*. In particular, a realtime class (RT) class and a timesharing (TS) class are included in SVR4. The scheduling classes are unified into a single priority scheduler by mapping each of them onto a range of global priorities, with timesharing processes mapped to the low priority range and realtime processes to the highest priority range. SVR4 also provides a set of commands for assigning processes to a class and controlling each class. These were used to assign processes for each experiment to the RT class, the TS class, or to a new scheduling class we developed as described later. In addition, for some experiments, controls specific to the scheduling class were used to modify their default behaviors.

In order to support the *video* continuous media application, an SBus I/O adaptor was constructed and added to the system that permits the decoding and digitization of analog video streams into a sequence of video frames. This video digitizing unit appears as a memory-mapped device in an application’s address space and allows a user-level application to acquire video frames, whose pixels can be color-space converted into RGB values, dithered to 8-bit depth, and displayed via the window system.

An effort was made to eliminate variations in the test environment to make the experiments repeatable. To this end, the testbed was disconnected from the network and restarted prior to each experimental run. In addition, to enable a realistic and repeatable sequence of typed keystrokes for programs of the interactive class, a keyboard/mouse simulator was constructed and attached to the testbed workstation. This device is capable of recording a sequence of keyboard and mouse inputs, and then replaying the sequence with the same timing characteristics.

4 Measurements

To evaluate a system’s performance, a means of measuring the system’s operation is needed that encompasses all of the activities in all of the applications. However, the measure of quality of an application’s performance is different for each class of application. To deliver the desired performance on interactive activities, the system should minimize the average and variance of time between user input and system response to a level that is faster than that which a human can readily detect. This means that for simple tasks such as typing, cursor motion, or mouse selection, system response time should be less than 50-150 milliseconds [11]. To deliver peak performance on display-oriented continuous media activities, the system should minimize the difference between the average display rate and the desired display rate, while also minimizing the variance of the display rate. In particular, uncertainty is worse than latency; users would rather have a 10 frames per second (fps) constant frame rate as opposed to a frame rate that varied noticeably from 2 fps to 30 fps with a mean of 15 fps [13]. To deliver good performance on batch activities, the system should strive to minimize the difference between the actual time of completion and the minimum time required for completion as defined by the case when the whole machine is dedicated to the given activity. In other words, if a *make* takes 10 minutes to complete on an unloaded system, the user would like the *make* to take $10\times(1+\delta)$ minutes, where δ is as small as possible, to complete even when there are other activities running on the system.

Because the relative value of each application to a user is subjective and application performance is measured in many different ways (i.e. interactive character latency verses video frame rate), no single figure-of-merit can be derived to compare test results. That is, any calculation resulting in a single value would require an assignment of weights and conversion factors to each measurement to account for the relative values of the applications and the different units of measurement. Since any such arbitrary assignment is suspect and is likely to obscure significant information, the outcome of each test is presented as a *value contour*. In a value contour, the achieved performance on each measurement is charted relative to a normative *baseline* value. If a single figure-of-merit is desired, it can be derived by assigning weights appropriate to the relative value of each application to the contour data.

Using value contours based on the mean and standard deviation of characteristic execution times, we capture the essential quality metric for each application class. The measured characteristic and baseline values are shown in Table 1 for each of the applications. To obtain these baseline values, each application was run in isolation on an otherwise quiescent workstation. Note, therefore, that when multiple applications are run simultaneously, it is not generally possible for all of them to reach 100% of the baseline value. The data from the experiments described in this paper, obtained from running these applications simultaneously, is shown in Table 2.

.

Application	**Measurement**	**Mean**	**Std. Dev.**
Typing	Latency between character arrival and rendering to frame buffer	38.5 msec.	15.7 msec.
Video	Time between display of successive frames	112 msec.	9.75 msec.
Compute	Time to execute one loop iteration	149 msec.	6.79 msec.

Table 1 Application Baseline Values

Application / Scheduling Class				Typing		Video		Compute	
X	T	V	C	χ (msec)	σ (msec)	χ (msec)	σ (msec)	χ (msec)	σ (msec)
TS	TS	TS	TS	42.9e+3	23.8e+3	2.78e+3	9.30e+3	150	16.0
TS+20	TS	TS	TS-20	49.6	26.4	117	17.9	3.91e+3	699
TS+20	TS	TS-5	TS-20	41.8	17.9	529	1.43e+3	189	279
TS+20	TS	TS-10	TS-20	44.0	18.5	174	619	412	896
TS	TS	RT	TS	—	—	1.10e+3	4.81e+3	243	415
RT	TS	TS	TS	26.4e+3	14.4e+3	4.23e+3	9.35e+3	150	22.9
RT-	TS	RT+	TS	—	—	142	260	—	—
RT+	TS	RT-	TS	42.0e+3	32.9e+3	112	8.09	8.04e+3	2.87e+3
New	New	New	New	46.0	19.1	177	48.3	496	114

Legend			
X	The X Window System server	TS	SVR4 TS (timesharing class)
T	The *typing* application	TS±*n*	SVR4 TS with nice of ±*n*
V	The *video* application	RT	SVR4 RT (realtime class)
C	The *compute* application	RT+	SVR4 RT with higher priority
χ	Mean	RT−	SVR4 RT with lower priority
σ	Standard Deviation	New	New scheduling class
— : Application did not complete measured operation			

Table 2 Individual Experiment Results

Figure 2 presents a set of value contours derived from this data. In each contour, the first two bars, labeled 'T_χ' and 'T_σ', represent the mean and standard deviation, respectively, for *typing* character latency. These values are normalized to the baseline values such that a full size bar represents a mean or standard deviation of latency as small as on an otherwise idle system (i.e. a taller bar represents better performance). Similarly, the bars labeled 'V_χ' and 'V_σ', represent the normalized mean and standard deviation of the time between display of successive frames for *video*. Finally, the bars labeled 'C_χ' and 'C_σ', represent the normalized mean and standard deviation of the time taken by one iteration of *compute*. The following section provides a description of the scenarios represented by each and an analysis of these results.

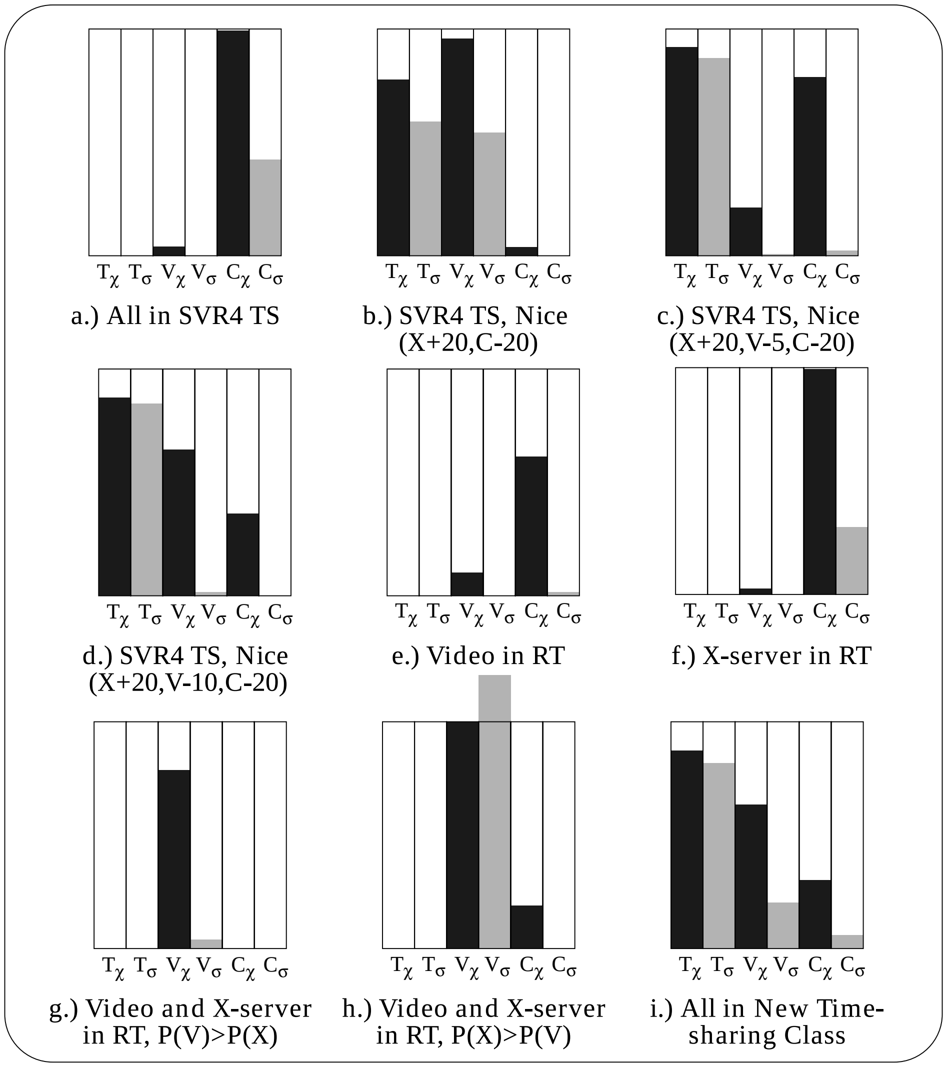


Figure 2 Application Value Contours

5 Interpretation of Results

It is expected that, in a well-behaved system, concurrent applications should all make some progress in their computation. That is, the running of an application by a user indicates some residual value for it. Therefore, no one application should be able to prevent others from running in absence of overt action by a user indicating this is the desired behavior. In addition, there should be no cases in which the system fails to respond to operator input; otherwise, control over the system is lost. Finally, users should be able to exercise a wide range of influence over the system's behaviors using a stable and predictable control mechanism.

The results of these experiments indicate that the standard SVR4 scheduling system often violates these objectives. The straightforward approach to adding multimedia applications to an SVR4-based workstation results, at best, in a low degree of value being provided to the users, and serious pathological behavior in the worst case. The following sections describe the test results for the SVR4 timesharing class alone, the SVR4 timesharing and realtime classes together, and a new implementation of the timesharing class.

5.1 SVR4 Timesharing Class

The first thing a typical user would do is to simply run the chosen set of applications, which, by default, associates all applications with the timesharing (TS) scheduling class. Doing this results in a pathological condition where the window system no longer accepts input events from the mouse or keyboard, causing the interactive application to freeze and the continuous media application to stop displaying frames of video. In fact, this pathology is so complete that attempts to stop the processes by typing commands in a shell (i.e. command interpreter) window prove futile, because the shell itself is not permitted to run.

The value contour for this scenario is shown in Figure 2a, and illustrates that all of the applications, with the exception of the batch job, contribute a relatively small amount to the total delivered value. This is due to the fact that the batch application forks many small programs to perform work, and then waits for them to finish. Because the batch application sleeps to wait for each child process to complete, the TS scheduling class identifies it as an I/O-intensive "interactive" job and provides it with repeated priority boosts for sleeping. As a result, the batch application quickly moves to the highest timesharing priority value and remains there for the remainder of the experimental run.

An added effect occurs when the window server develops a backlog of outstanding service requests. As it works down this queue of outstanding commands, the TS scheduling class identifies the window server as CPU-intensive and lowers its priority. At the same time, because it sleeps in the process of obtaining new video frames, *video* is assigned a higher priority, allowing it to run and thereby generate additional traffic for the window server. As a result, the quality of the video being displayed is poor because the window system is not able to execute to process the frames fast enough. Worse yet, *typing* exhibits an average delay of more than 42 seconds from receiving a character to having it displayed, as opposed to the baseline value of 39 milliseconds. The interactive application suffers a degradation of three orders of magnitude because the window server, which must execute in order to render the character's pixels to the frame buffer, is not scheduled to run frequently enough to work its way through its growing backlog of commands. Moreover, due to the design of the standard SVR4 TS class, it can often take tens of seconds for the priority of a penalized process to recover to the point at which it can actually run. This augments the effect of the improper processor scheduling decisions and contributes to the poor overall performance of the system.

In an attempt to deal with this problem, the system's administrative controls were used to change the TS priorities of the window system and the applications. These user priorities are used by the TS scheduler to modify the actual scheduling priorities. These controls correspond roughly with the traditional UNIX *nice* values. In one case, the user priority of the window system was elevated to the maximum possible level (+20), while the user priority of *video* was depressed to the minimum possible level (−20), as shown in Figure 2b. This had the effect of improving *video*'s performance, but the latency of *typing* became more variable and *compute* barely ran. In an attempt to fix this, the user priority of *video* was degraded modestly (−5), resulting in the contour in Figure 2c. This shows how very small changes in these controls can lead to large and unpredictable effects. Finally, Figure 2d illustrates the result of *video* receiving a medium amount of degradation (−10). The achieved mean values of all applications are relatively high, but the variance in frame rate for *video* is unacceptably high. Note also the counterintuitive result that *video* performs better in this scenario then in Figure 2e, even though the scheduler controls indicated a lower importance for *video*.

Although the use of user priority adjustments could alleviate the pathological condition inherent in the SVR4 TS scheduling class, this approach is not effective in general (e.g., with multiple, independent applications). That is, it can take a great deal of experimentation in order to find a set of control values that work well, and the settings might only work for that exact application mix. In addition, this approach severely degrades the performance of *video*, resulting in highly variable display rates.

5.2 SVR4 Timesharing and Realtime Classes

Although SVR4 UNIX also provides so-called "realtime" facilities, the assignment of different tasks to the realtime (RT) scheduling class yielded equally unsatisfactory results. Since *video* best fits the notion of what a realtime application is, the obvious first step for using the RT class is to assign *video* to it. However, when this is done, the system again ceases to accept input events from the mouse or keyboard and the video again degrades severely. This is due to the fact that any ready task in the RT class takes precedence over any TS task. Since *video* is almost always active, tasks in the TS class are hardly ever allowed to execute — in fact, shell programs are not even permitted to run, so a user cannot even attempt to stop such a "realtime" application. Once again, the quality of the video being displayed is poor because the window system is not able to execute to process the frames sent to it by the continuous media application. Again, the system delivers low overall value for any choice of value assignments, as shown in Figure 2e.

Alternatively, the window system could be associated with the RT class, with all of the applications remaining in the TS class. Although in such a case, the window system related activities (e.g., mouse tracking) perform well, the basic TS scheduling system pathology allows the batch job to monopolize the processor. As a result, none of the other applications can achieve even a small fraction of their possible value, as illustrated in Figure 2f.

Another attempt to provide a high degree of value to the user involves placing both *video* and the window system in the RT class, and having all applications remain in the TS class. In this case, the system executes *video* to the complete exclusion of all other processing. That is, neither *typing* nor *compute* are permitted to run at all, and it is not possible to type commands into the system's shell windows. In fact, basic kernel services such as the process swapping, flushing dirty pages to disk, and releasing freed kernel memory are inhibited. The reason for this behavior is that *video* and the window server consume essentially all of the system's processor cycles, and realtime processes take precedence over all "system" and timesharing processes. This is because the RT scheduler uses a strict priority policy, and no processes from other scheduling classes are permitted to run while there are ready processes in the RT class.

Figure 2f and Figure 2g show the results that are derived from placing the window system at a lower and at a higher RT priority than *video*, respectively. While neither case delivers acceptable results, the first case (i.e., with the window server's priority below *video*) was particularly bad because *video* did not leave sufficient time for the window server to process its requests. Note also, that in Figure 2h, *video* had less variance than in the baseline measurements. This is due to the strict priority scheduling discipline; processes in the RT class run in preference to all other processes, including system daemons.

Finally, we note that placing interactive applications in the RT class in order to improve their performance would also be ineffective unless the window server were placed in the RT class. Even then, proper operation is not assured because basic system services can be prevented from functioning due to resource demands in the higher priority realtime class. For example, when the X window server, *typing*, and *video* are run in the RT class, with priorities P(X)>P(*typing*)>P(*video*), *typing* unexpectedly performs more than three times worse than its baseline because it relies on streams I/O services [1] for character input processing. Because the streams processing is not done in the RT class, it is deferred in favor of the applications in RT, which consume virtually all of the CPU cycles.

5.3 New Timesharing Class

A new timesharing scheduling class was developed in order to correct the problems demonstrated in these experimental runs. In particular, the modified version removes the anomalies of identifying batch jobs as interactive, and vice versa. In addition, it attempts to ensure that each process that can run is given the opportunity to make steady progress in its computation, while retaining a bias in favor of interactive processes. Finally, it reduces the feedback interval over which CPU behavior is monitored and penalties and rewards given. The timesharing scheduling class contained in Sun's Solaris 2.3 operating system is based on this work.

The results of the default use of this class for all applications and the window server process are given by Figure 2i. As can be seen, this delivers significantly better results for the continuous media and interactive applications than any combination of the standard SVR4 scheduling classes. It should also be noted that this scheduling policy achieves this level of performance without significantly starving the batch application, which still receives approximately 30% of the available CPU time.

Additional tests were performed by adjusting user priorities and by combining this new scheduling class with the SVR4 RT class (as was done with SVR4 TS class). However, with the exception of the cases where there was sufficient load in the RT class to consume all CPU cycles and starve the new scheduling class, this resulted in no pathologies and showed a direct and predictable relationship between user priorities and application performance.

6 Conclusions and Future Work

Through trial and error, it may be possible to find a particular combination of priorities and scheduling class assignments to make the SVR4 scheduling pathologies go away. However, such a solution would be extremely fragile and would require discovering new settings for any change in the mix of applications. In fact, these problems have been induced in many instances with different applications and conditions than those described here. For example, the continuous media application by itself can freeze the system when a user simply uses a popup menu. Our new timesharing scheduling class eliminates these pathologies and provides default resource management behavior that favors interactive applications while not overly penalizing others.

Current workstation operating systems, typified by SVR4 UNIX, evolved from the much different environment of large-scale, multi-user, timesharing systems. These systems attempt to be fair to all applications while maximizing total system throughput. As a result, a user (or system administrator) has only limited control over UNIX operating system resource management decisions.

Without such control it is not possible to provide the full range of behaviors that might be desired of multimedia applications. For example, providing uniform rates of audio and video presentation, where variance in the delivery rate is minimized, may be more important to some applications than others. Knowledge of the "slack" available in such computations can lead to more effective resource utilization. In addition, when the system is overloaded with continuous media applications, a way of identifying applications of lesser or greater importance to the users can allow the system to automatically perform service trade-offs rather than forcing it to degrade all applications equally at best, or randomly at worst. Armed with such information, the system can manage its resources in such a way as to maximize the total value delivered to the end user. Towards this end, we are creating a new scheduling framework, based on Time-Driven Resource Management [6, 8, 9], that provides the flexible control and delivered performance required for multimedia applications.

Finally, note that the existence of the strict-priority realtime scheduling class in standard SVR4 in no way allows a user to effectively deal with these types of problems. In addition, it opens the very real possibility of runaway applications that consume all CPU resources and effectively prevent a user or system administrator from regaining control without rebooting the system.

7 Acknowledgments

Monica Lam provided many insightful suggestions, especially during the formative stages of this work. This research was supported in part by an NSF Young Investigator Award and Sun Microsystems Laboratories, Inc.

[‡]UNIX is a trademark of UNIX System Laboratories.

8 References

1. AT&T: *UNIX System V Release 4 Internals Student Guide*, Vol. I, Unit 2.4.2., AT&T, 1990.

2. M. J. Bac*h: The Design of the UNIX Operating System*, Prentice Hall Inc., 1986

3. J. Bonwick: “Kernel Tracing in SunOS 5.0,” in progress.

4. S. Evans, K. Clarke, D. Singleton, B. Smaalders: “Optimizing Unix Resource Scheduling for User Interaction,” USENIX Summer 1993, Cincinnati, Ohio.

5. J. R. Eykholt, S. R. Kleiman, S. Barton, R. Faulkner, et. al.: “Beyond Multiprocessing...Multithreading the SunOS Kernel,” USENIX Summer 1992, San Antonio, Texas.

6. J. G. Hanko, E. M. Kuerner, J. D. Northcutt, and G. A. Wall: “Workstation Support for Time-Critical Applications”, Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November, 1991.

7. S. Khanna, M. Sebree, J. Zolnowsky: “Realtime Scheduling in SunOS 5.0,” USENIX Winter 1992, San Francisco, California.

8. J. D. Northcutt, J. G. Hanko, and G. A. Wall: “A New Framework for Processor Scheduling,” in progress.

9. J. D. Northcutt: *The Alpha Operating System: Requirements and Rationale*, Archons Project Technical Report #88011, Department of Computer Science, Carnegie-Mellon University, January 1988

10. R. W. Scheifler and J. Gettys: “The X Window System,” *ACM Transactions on Graphics*, 5(2), April, 1986.

11. B. Shneiderman: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 2nd ed., Addison-Wesley, 1992.

12. G. A. Wall, J. G. Hanko, and J. D. Northcutt: “Bus Bandwidth Management in a High Resolution Video Workstation,” Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, November, 1992.

13. T. Winograd: personal communication, March 1993.