Workstation Video Playback Performance with Competitive Process Load

Kevin Fall[1], Joseph Pasquale[1] and Steven McCanne[2]

[1] University of California, San Diego, CA 92093–0114, USA
[2] Lawrence Berkeley Laboratory, Berkeley, CA 94720, USA

Abstract. While many researchers believe that multimedia applications are best managed with hard, real-time scheduling mechanisms, models based on application-level adaptation with relaxed scheduling constraints are gaining acceptance. We analyze an existing video conferencing application, which was designed without explicit support for CPU resource management, and propose modifications to its architecture to support CPU load adaptation. Inter-frame display times (IDT) for a 30 frames/sec video segment are analyzed for increasing multiprocessing loads. As expected, the IDT variance (and mean) increases markedly with load, especially beyond saturation. We show that this display jitter can be significantly reduced by gracefully adapting the application's load requirements to match the available CPU resources.

1 Introduction

Modern workstations equipped with multimedia hardware and running conventional non-real-time operating systems have been shown to offer acceptable video playback performance under light multiprocessing loads. When required to compete with other processes for CPU resources, continuous media playback continuity can be adversely affected.

Improving playback continuity is generally accomplished by employing reservation-based CPU scheduling, or by shedding load when resources are in short supply. Several real-time schemes employ schedulers which require performance parameters to be specified prior to program execution [1, 2, 3]. Other real-time systems strive to alleviate the user from preselecting resource requirements, and instead rely on *adaptive resource management* (ARM) [4, 5]. In ARM, applications self-adapt over time in order to match their resource demands to the system's resource availability. Although ARM was originally proposed for real-time systems, it can also improve system performance on non-real-time scheduled systems.

We define multiple processes executing to accomplish a common task *cooperating processes*. Unrelated (interfering) processes are called *competitive processes*. Generally, the traditional scheduler (the UNIX scheduler in our case) provides poor playback continuity to processes unable to modulate their resource utilization whether they are cooperative or competitive. Even when real-time kernel support is available, specifying the parameters needed to ensure cooperative processes are scheduled at appropriate times is difficult [6].

When competitive processes are present, an application may fail to deliver its media stream at a sufficiently fast rate or at a sufficiently regular rate to maintain playback continuity. In this overloaded state, applications can resort to *load shedding* to reduce their CPU resource requirements. It has been suggested that load shedding may be combined with application-implemented adaptive control to provide good continuous media performance without the need for user-specified resource requirements [7]. Such techniques have already been demonstrated to be effective for handling network congestion [8, 9].

This paper explores the effects of competitive process load upon the otherwise smooth operation of a multi-process video playback application on a non-real-time operating system. We examine the playback performance of a video conferencing application that was designed without explicit support for CPU resource adaptation and which consequently performs poorly under competitive process load. To alleviate this problem, we propose a modified design and present measurements that demonstrate the efficacy of our approach.

2 Current Architecture

To investigate the effect of competitive process load on the operation of a continuous media application, we instrumented the UCB/LBL video conferencing application, *vic* [10]. The basic architecture is illustrated

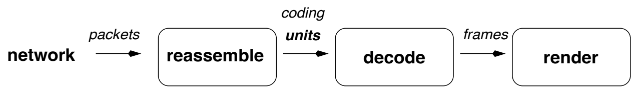

Fig. 1. Receive data path processing in `vic`.

in Figure 1. Packets containing encoded video arrive from the network and are reassembled into encoding-specific framing units, which are decoded by a software or hardware codec. The decoded frames are then rendered and displayed. In the current implementation, if the receive data path cannot keep up with the incoming data stream, packets are dropped because of input network buffer overflows, leading to an erratic degradation of performance in the presence of competitive load.

The configuration under study consists of the three processes. The `vic` process executes in tandem with a video device server (`jvdriver`) and the X server. The platform under test was a DEC Alpha 3000/M500 running under DEC OSF/1 v. 1.3. DEC's "JVideo" hardware codec, managed by the `jvdriver` process, performs Motion-JPEG decompression at a target rate of 30 frames per second. The JVideo hardware decompresses, scales, and dithers each frame. Compressed and uncompressed frames are both maintained in shared memory buffers to reduce memory-to-memory copies.

To isolate the CPU scheduling effects from network induced jitter, we used an early version of `vic` which allowed compressed frames to be played back from local memory. A periodic timer, driven off wall clock time, schedules frames to be decompressed every 33ms. When `vic` gets behind, it attempts to catch up by running faster than 30Hz. If it gets too far behind (200ms), it catches up by resetting the frame clock, resulting in a burst of lost frames and a visible discontinuity in the playback video.

`Vic` operates by scheduling a software timer every 33ms. When the timer expires, the next JPEG frame is queued to `jvdriver`, which programs the JVideo hardware to decompress (and dither) the frame. The compressed frame is transferred from an application shared memory buffer directly to the hardware by DMA. Upon decompression, the decompressed frame is transferred to a seperate application buffer in a similar fashion. `Jvdriver` signals `vic` each time a frame has been decompressed, at which time `vic` displays the frame by copying the image data into the frame buffer (via the X server). Note that the uncompressed image buffer is shared by `vic`, the X server, `jvdriver`, and the hardware.

Our data was collected by instrumenting `vic` to log a timestamp each time the X server finished displaying a frame. From these absolute frame times, we compute the inter-frame display times (IDTs). The variance of IDT samples represents the irregularity or *jitter* of frame buffer display updates and provides a quantitative measure of playback continuity. We used a 16.8MByte Motion-JPEG trace file that was read into memory at start up. The file fit entirely within physical memory and required no subsequent disk access.

3 An Adaptive Architecture

For smooth media playback, the IDTs should remain nearly constant; that is, the IDT sample variance should be close to zero. Not surprisingly, we found the IDT variance to be strongly correlated to the level of compute-bound competitive load. As more processes compete for the CPU, the likelihood that `vic` gets to run when its frame timer expires becomes lower. Moreover, when it does run, it further increases the IDT variance (and hence display update jitter) by trying to catch up as fast as possible.

In order to maintain smooth media playback, the application must match its CPU demand to current availability. We pursue this idea with the modified architecture in Figure 2, which has been similarly proposed in [7]. Here, a load monitoring agent determines when CPU resources become scarce and induces the application to shed load. Within `vic`, load shedding could occur by rendering frames at a rate below the source rate. A stateful decoder might still need to decode every frame that arrives, but not every frame need be dithered and displayed (which often accounts for the majority of the computation). Alternatively, parameters to a compute-scalable decoder might be altered (for example, by employing arithmetic approximations for faster decoding at lower quality). Later, when resources become plentiful, load can be increased. Exactly how one measures the load and how aggressively one reacts is a difficult problem. The overall structure corresponds to a non-linear, time-varying feedback-control system.

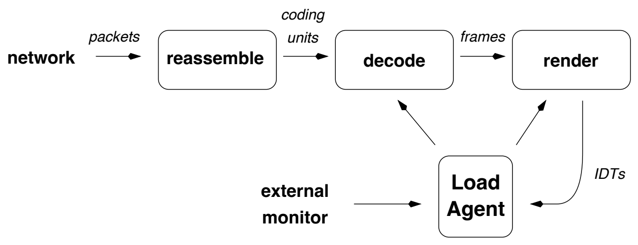

Fig. 2. Receive data path enhanced with load agent.

Designing a general control algorithm that reacts quickly without oscillating is beyond the scope of this paper and will be investigated in the future. Instead, we focus on a simple control algorithm that we adjust by hand to demonstrate the viability of the model in a particular case. In Motion-JPEG, compressed frames are independent of each other, so we can very simply adjust our induced load by dropping some number of frames at the input (i.e. before they are decompressed). For example, M out of N spaced frames ($M < N$) can be dropped uniformly to gracefully degrade the playback performance and reduce load. We can determine the fraction M/N by continually reducing it until a satisfactorily low IDT variance is achieved. In other words, we infer load by monitoring IDTs and shed load with frame decimation. Uniform frame dropping distributes load shedding equally over time, reducing perceived playback discontinuities. Since IDTs will not indicate when the CPU becomes underutilized, some other mechanism must be used (labeled "external monitor" in the figure).

We call this approach *early discard load shedding* (EDLS) because it discards frames deterministically at the input, instead of relying implicitly on input buffer overflows to shed load. We term the latter method *drop-on-overload*, because load is shed well *after* the system is heavily saturated. The distinction between drop-on-overload and EDLS is analogous to that between drop-tail and random-early drop (RED) gateways [11] in packet networks. Here, drop-tail routers react to load by dropping packets only after their queues are full, while RED gateways attempt to react incipient congestion before the router becomes overloaded.

4 Results

Figure 3 shows various statistics as a function of multiprocessing load. A load of n indicates that n compute-bound processes are competing for CPU time with `vic`. The graph indicates results for both the drop-on-overload, as well as EDLS. Statistics include the mean, median, and standard deviation of IDT. All values are observed results from our experiments.

We define the *saturation point* to be the point at which the standard deviation exceeds the median IDT. This point occurs at a CPU load of approximately 3.8. Below the saturation point, the playback application functions well enough to provide qualitatively continuous video, but exhibits observable discontinuities above this point.

The curves to the left of the saturation point depict the operation of `vic` under drop-on-overload. The three additional curves to the right of the saturation point depict the same statistics computed for `vic` using EDLS. With drop-on-overload, the standard deviation and mean of the IDT grows with increasing process load. When EDLS is used, the IDT deviation remains low and relatively flat as compared with its mean and median. The small deviation with EDLS results in improved qualitative video playback continuity beyond the saturation point as compared with the drop-on-overload policy.

As can be seen from the graph, the median IDT remains constant across load for the drop-on-overload case and grows with the mean for the EDLS case. The median predicts the most common IDT more accurately than the mean for high variance. For drop-on-overload, the standard deviation grows noticeably beyond load level 3, but decreases with increasing median in the EDLS case. This effect is due to the way EDLS performs decimation based on load. Increasing load causes a reduction in frame rate requested at the display process. The reduction in frame rate gives rise to longer inter-frame times with less dispersion, explaining the similar curves for mean and median in the EDLS case.

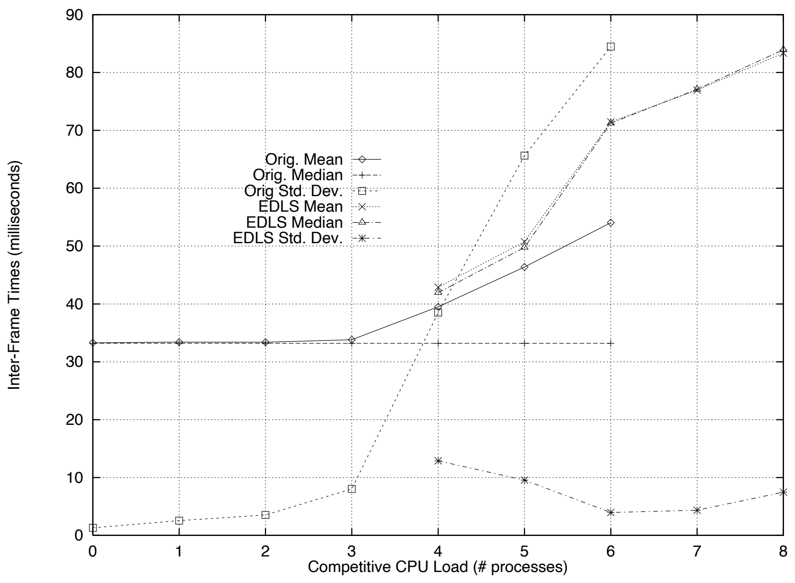

Fig. 3. Comparison of Drop-on-Overload and EDLS

5 Conclusion

In this paper, we explored the effects of the traditional Unix scheduler on an application that requires periodic execution to display continuous media. We proposed a generic framework, based on the existing software architecture in `vic`, for CPU load adaptation. By considering a special case of the general architecture, namely EDLS of Motion-JPEG frames using IDT sample variance, we demonstrated the viability of the approach.

We defined a *saturation point* with respect to competitive process load and showed that beyond this point, playback performance under the drop-on-overload policy suffers dramatically. By using the IDT sample variance to detect saturation, EDLS reacts to overload and adjusts the playback rate to gracefully degrade performance in the presence of competitive load.

References

1. H. Tokuda and T. Kitayama, "Dynamic QOS Control Based on Real-Time Threads," *Proc. 4th Intl. Workshop on Network and OS Support for Digital Audio and Video*, 1993.
2. D. Anderson, "Metascheduling for Continuous Media," *TOCS*, vol. 11, pp. 226–252, Aug 1993.
3. K. Jeffay, D. L. Stone, T. Talley, and S. F. D, "Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks," *Proc. 3rd Intl Workshop on Network and Operating System Support for Digital Audio and Video*, Nov 1992.
4. M. Jones, "Adaptive Real-Time Resource Management Supporting Modular Composition of Digital Multimedia Services," *Proc. 4th Intl. Workshop on Network and OS Support for Digital Audio and Video*, 1993.
5. J. D. Northcutt and E. M. Kuerner, "System Support for Time-Critical Media Applications," *Sun Microsystems Internal Research Notes*, no. SMLI-92-0077, 1990.
6. J. Nieh, J. Hanko, D. Northcutt, and G. Wall, "SVR4 UNIX Scheduler Unacceptable for Multimedia Applications," *Proc. 4th Intl. Workshop on Network and OS Support for Digital Audio and Video*, 1993.
7. C. Compton and D. Tennenhouse, "Collaborative Load Shedding for Media-Based Applications," *International Conference on Multimedia Computing and Systems*, May 1994.
8. H. Kanakia, P. Mishra, and A. Reibman, "An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport," *Proc. SIGCOMM '93*, Sep 1993.
9. V. Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM '88*, Aug 1988.
10. S. McCanne and V. Jacobson, *VIC: Video Conference*. U.C. Berkeley and Lawrence Berkeley Laboratory. Software available via ftp://ftp.ee.lbl.gov/conferencing/vic.
11. S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug. 1993.