Meeting CPU Constraints by Delaying Playout of Multimedia Tasks

Balaji Raman Samarjit Chakraborty Wei Tsang Ooi
Department of Computer Science
National University of Singapore
{ramanbal, samarjit, ooiwt}@comp.nus.edu.sg

ABSTRACT

Multimedia applications today constitute a significant fraction of the workload running on portable devices such as mobile phones, PDAs and MP3 players. However, the processors in such devices are usually not powerful enough to support multiple concurrently executing multimedia tasks. In this context, different processor scheduling algorithms have attracted a lot of attention. This paper attempts to address the CPU constraint problem from a different perspective. It is based on the observation that by increasing the playout delay of a multimedia task, the minimum processor frequency required to run the task decreases. This is due to the high data-dependent variability in the execution requirements of multimedia tasks. We also present a framework, using which it is possible to compute the minimum processor frequency corresponding to any playout delay. Given a set of concurrently executing multimedia tasks, using our framework it is possible to compute the playout delays for each of these tasks, such that the sum of their corresponding processor cycle requirements do not exceed the maximum frequency supported by the processor.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Real-Time and Embedded Systems

General Terms

Theory, Measurement, Performance, Experimentation

Keywords

Scheduling Multimedia Tasks, Playout Delay, Buffering

1. INTRODUCTION

The processor cycles of devices like PDAs, mobile phones and portable audio/video players are today mostly spent in running multimedia-rich applications. However, the processors in such devices are significantly slower than those found in even low-end desktops. For example, many high-end PDAs today have 400 MHz Intel XScale processors, in contrast to 2.5 - 3 GHz processors commonly found in desktops. As a result, there is a significant gap between the processing resources available in these portable devices and the requirements of applications that users would want to run on them. Although a lot of work in the domain of processor scheduling—especially in the context of multimedia applications—can be used to address this problem, in this paper we look at it from a different perspective. We note that by increasing the playout delay of a multimedia task, there is a decrease in the minimum processor frequency with which it needs to be run. Typically the playout delay of a multimedia task is chosen independently of the other tasks running on the processor. When multiple such tasks are to be run concurrently, it might so happen that the sum of their processor cycle requirements exceed the maximum frequency with which the processor can be clocked. However, in such cases, by appropriately delaying the playout of each of these tasks (which depends on the load on the processor) it might be possible to support all of them.

Most multimedia applications exhibit a high degree of data-dependent variability in their execution requirements. In other words, when such an application processes a stream of data items (e.g. macroblocks or frames in the case of MPEG decoding), the number of processor cycles required to process each data item is highly variable. The ratio of the worst-case and the average load on a processor running such an application can be as high as a factor of 10 [15]. The playout rate associated with the application (e.g. the rate at which decoded frames are displayed by the output device in the case of an MPEG decoder) imposes certain real-time constraints on it. When the playout delay is negligible, such constraints translate to an upper bound on the amount of time that can be spent in processing each data item. Since the number of processor cycles required by each data item is variable, the minimum processor frequency is determined by the item which requires the maximum number of processor cycles. When the playout of the application is delayed (i.e. the processed data items are buffered before being played out), the minimum required processor frequency decreases. For any given delay, the "amount" of decrease is proportional to the variability in the execution requirement of the stream. With a sufficiently large playout delay, the minimum required processor frequency corresponds to the average processor cycle requirement per data item.

Further, many multimedia tasks have variable input-output rates, i.e. the number of input data items consumed to pro-

NOSSDAV'05, June 13–14, 2005, Stevenson, Washington, USA.

duce one processed data item at the output is variable. For example, the variable length decoding task in an MPEG decoder consumes a variable number of bits to produce one partially decoded macroblock. This provides additional possibility for reducing the required processor frequency by delaying the playout of such tasks.

Following the above observations, in this paper we present a framework using which it is possible to precisely compute the minimum processor frequency required by a task for any given playout delay. We also demonstrate how to appropriately choose the playout delays of multiple concurrently executing tasks such that their minimum processor cycle requirements can be satisfied by a given processor.

Organization of the paper: The rest of the paper is organized as follows. In the next section we present a motivating example to illustrate the key idea behind this work. In Section 3 we examine some related previous work. This is followed by our proposed framework in Section 4 and its experimental evaluation in Section 5. Finally, in Section 6 we conclude by outlining some directions for future work.

2. MOTIVATING EXAMPLE

Playout Delay (in seconds)	**Minimum Processor Frequency** (in MHz)		
	MPEG2	MP3	Total
0.10	1356	346.2	1702.2
0.12	1033	327.5	1360.5
0.14	664.8	318.4	983.2
0.16	347.3	317.1	664.4
0.18	344.6	315.6	660.2
0.20	342.0	314.9	656.8

Table 1: Processor frequency requirements for an MPEG-2 and an MP3 stream.

Let us consider a portable device with a processor core running at 750 MHz. Assume that there are some tasks already running on the device and they require a small fraction of the available processor bandwidth, say 50 MHz. Now consider a scenario where the scheduler has to synchronize and continuously play an MPEG-2 stream and an associated MP3 audio clip. As mentioned earlier, the main observation that motivates this work is that an increased playout delay can counter the variability present in multimedia workloads, and consequently reduce the minimum processor frequency required to process a multimedia stream. Now, presume that the scheduler has computed the processor frequency values for several playout delays, as shown in Table 1. This table shows the processor frequency required by a video clip `mobile.mv2` (see Table 2) and an audio clip `amazing.mp3` (see Table 3). From this table the scheduler can now choose a common playout delay of 0.18 seconds and continuously play the video and the audio at a frequency of 660.2 MHz. Note that we still have around 40 MHz of remaining processor bandwidth. This can be used to accommodate any potential, computationally less intensive tasks that could pop up while playing the video.

In summary, the scheduler chose a playout delay with which it is possible to concurrently run both, the audio and the video decoding tasks. With a smaller delay (e.g. 0.1 seconds), it would not be possible to meet the real-time playout constraints associated with these tasks using a 750 MHz processor. Further, it may also be noted that:

- There was no hindrance to the tasks currently running on the processor.
- The processor cycles were effectively utilized. As discussed in Section 1, by sufficiently delaying the playout of a task, it is possible to run the task at a frequency that corresponds to the average processor cycle requirement of the stream. Clearly, the processor cannot be run at a frequency lower than this.
- The multimedia tasks were not run at the cost of shunning any impending tasks.

It is also worth noting that for voltage/frequency-scalable processors the above scheduler can appropriately scale the processor frequency depending on the load on the processor and battery-life desired by the user. However, we do not explore this possibility here; this is an interesting research direction that we hope to pursue in future.

In Section 4 of this paper we present a framework using which Table 1 may be computed for any given multimedia application.

3. RELATED WORK

In this section we discuss some of the previously proposed techniques for estimating processor cycle requirements of multimedia tasks, with the aim of either improving CPU utilization or minimizing energy consumption. We also point out the major differences between these techniques and our proposed framework.

All dynamic frequency or voltage scaling algorithms rely on the fact that lowering the processor's clock frequency and/or supply voltage reduces its power consumption. However, accurately predicting the variation in the workload generated by a multimedia task, so that the processor's frequency can be changed accordingly, is a difficult problem [4, 16]. In [16], the resource requirement for the current workload in the case of MPEG decoding is predicted from the frame drop and delay encountered during the immediately previous time interval. Further, the time required to decode any MPEG frame is predicted using a frame classification technique. On the other hand, in [4], the decoding process is classified into two parts: frame dependent and frame independent decoding. The decoding time for both these parts are estimated and the processor's voltage is scaled accordingly. In contrast to these approaches, our work relies on an offline analysis of a multimedia stream to accurately characterize the variability in the workload generated by it. It does not rely on any runtime prediction of the processor cycle requirements of the stream.

A number of papers also exploit buffering techniques to smooth out the variability in multimedia workloads, so that dynamic power management techniques can be used [3, 8, 11, 12]. In [8], a job scheduler delays the execution of soft real-time tasks, by buffering the streams processed by these tasks, and uses the generated slack to process tasks with stringent real-time constraints. However, this technique does not provide any insights into what would be an optimal buffering time that would lead to energy savings and at the same time allow continuous playback of a multimedia task. Our framework, on the other hand, can be used to

identify an optimal playout delay, such that increasing this delay does not lead to further savings in processor cycle requirements and decreasing this delay significantly increases the processor cycle requirements of a task.

Lately, a number of energy aware scheduling techniques have been proposed in the literature (see [6, 19, 9] and the references therein). A few recent papers have also addressed the problem of energy savings in the specific context of multimedia applications [7, 17]. Our work differs from these in the following ways:

- Our framework applies to any kind of multimedia streaming application. In other words, it does not rely on the specific characteristics of the application.
- The analytical framework that we present can be applied to any level of granularity i.e. each data item in the stream can be a bit, a macroblock, or a frame.
- The framework is only concerned with computing the buffering time or the playout delay of each stream. It is independent of the scheduling policy used to schedule these streams. This framework is supposed to work at a level below the scheduler which would typically be used to schedule the multiple streams being processed by the device.
- As mentioned earlier, the framework that we present in this paper can be extended to the case of dynamic voltage-frequency scaling. However, we do not explore this option here.

In the domain of computer networks, several techniques have been proposed for exploiting the playout delay of multimedia applications in the context of link scheduling [13, 14]. For example, in [13], tight bounds on optimum average audio playout delay based on tradeoffs between per-packet loss and delay were estimated. Using these bounds, a history dependent adaptive packet playout delay adjustment algorithm was proposed. Our work is concerned with computing a fixed playout delay, rather than dynamically adjusting it at runtime. Further, our technique is more relevant in the context of playing stored audio and video. Hence, we did not exploit any network-related parameters such as packet loss and delay.

4. DELAY-FREQUENCY ESTIMATION

In this section we present our analytical framework to compute the minimum processor frequency in order to continuously playback a given media file, with a playout delay equal to d.

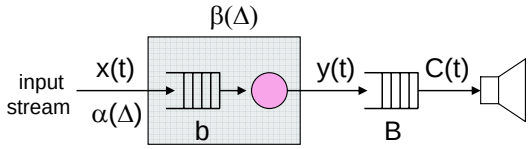
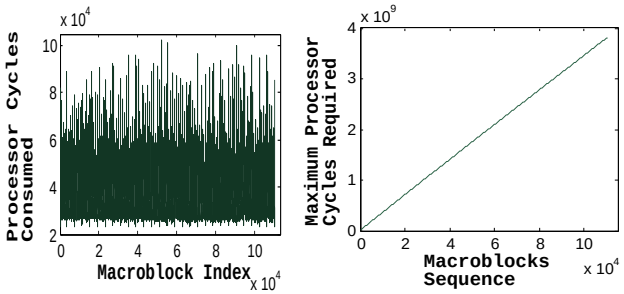
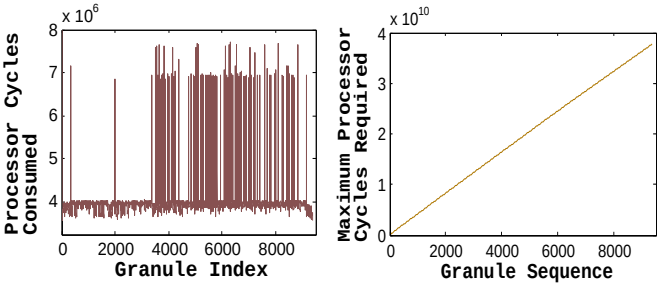


Figure 1: System model.

Figure 1 shows our system model, which consists of a processor with an internal buffer, a playout buffer and a playout (or output) device. After decoding the input stream, the processor writes the data in the playout buffer which is consumed by the playout device (e.g. a video display or a speaker) at a fixed rate.



(a) `mobile.m2v`: Processor cycles/macroblock and the corresponding $\gamma^u(t)$



(b) `amazing.mp3`: Processor cycles/granule and the corresponding $\gamma^u(t)$

Figure 2: Variation in the processor cycle requirement per stream object, for video and audio streams.

We assume that the input bit stream to be decoded is fed into the internal buffer at a constant rate of r bits/sec. Further, for the sake of simplicity, we will consider a stream to be made up of a sequence of *stream objects*. A stream object might be a macroblock in the case of video decoding or a granule in the case of audio decoding tasks. Now, given a media clip to be decoded, let $x(t)$ denote the number of stream objects arriving in the internal buffer over the time interval $[0, t]$. Due to the variability in the number of bits constituting a stream object, the function $x(t)$ varies with the media clip. We define two functions $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ to bound the variability in the arrival process of the stream objects into the internal buffer of the processor. These two functions are defined as:

$$\alpha^l(\Delta) \leq x(t+\Delta) - x(t) \leq \alpha^u(\Delta) \quad (1)$$

for all t and $\Delta \geq 0$, where $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denotes the minimum and maximum number of stream objects that can arrive in the internal buffer within *any* time interval of length Δ, respectively.

To compute $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$, we introduce two functions $\phi^l(k)$ and $\phi^u(k)$. The former denotes the minimum number of bits constituting *any* k consecutive stream objects in an audio bitstream, and the latter denotes the corresponding maximum number of bits. These two functions can be obtained by analyzing a number of media clips that are *representative* of the clips to be processed by the target decoder.

Given the functions $\phi^l(k)$ and $\phi^u(k)$, it is possible to compute the *pseudo-inverse* of these two functions, denoted by $\phi^{l^{-1}}(n)$ and $\phi^{u^{-1}}(n)$, where the argument n is the number of bits. $\phi^{l^{-1}}(n)$ and $\phi^{u^{-1}}(n)$ returns the maximum and minimum number of stream objects that can be constituted by n bits respectively. Since we assume the input bit stream

arrives in the internal buffer at a constant rate of r bits/sec, we have:

$$\alpha^l(\Delta) = \phi^{u^{-1}}(r\Delta) \text{ and } \alpha^u(\Delta) = \phi^{l^{-1}}(r\Delta)$$

Similarly, we can characterize the variability in the number of processor cycles required to process any stream object using two functions $\gamma^l(k)$ and $\gamma^u(k)$. Both these functions take the number of stream objects k as an argument. $\gamma^l(k)$ returns the minimum number of processor cycles required to process *any* k consecutive stream objects and $\gamma^u(k)$ returns the corresponding maximum number of processor cycles.

Finally, we assume that the playout buffer is readout by the output device at a constant rate of c stream objects/sec, after a playout delay (or buffering time) of d seconds. Let the function $C(t)$ be the number of stream objects readout by the output device over the time interval $[0, t]$, then obviously,

$$C(t) = \begin{cases} 0 & \text{if } t \leq d \\ c(t-d) & \text{if } t > d \end{cases} \quad (2)$$

Now, given the input bitrate r, the functions $\phi^l(k)$, $\phi^u(k)$, $\gamma^l(k)$ and $\gamma^u(k)$ characterizing the possible set of media clips to be decoded, and the function $C(t)$, we can compute the minimum processor frequency f to sustain the playout rate of c stream objects/sec. This is equivalent to requiring that the playout buffer never underflows. Let $y(t)$ denotes the total number of stream objects written into the playout buffer over the time interval $[0, t]$. Then the playout buffer underflow constraint is equivalent to requiring that $y(t) \geq C(t)$ for all $t \geq 0$.

Let the *service* provided by the processor at frequency f be represented by the function $\beta(\Delta)$. Similar to $\alpha^l(\Delta)$, $\beta(\Delta)$ represents the minimum number of stream objects that are guaranteed to be processed (if available in the internal buffer) within any time interval of length Δ. It can be shown that [10] $y(t) \geq (\alpha^l \otimes \beta)(t), \forall t \geq 0$, where $\otimes$ is the *min-plus convolution* operator[1]. Hence, for the constraint $y(t) \geq C(t), \forall t \geq 0$ to hold, it is sufficient that the following inequality holds:

$$(\alpha^l \otimes \beta)(t) \geq C(t), \ \forall t \geq 0 \quad (3)$$

It is known from the duality between $\otimes$ and $\oslash$, that for any three functions f, g and h, $h \geq f \oslash g$ if and only if $g \otimes h \geq f$ (see [2] for further details), where $\oslash$ is the *min-plus deconvolution* operator[2]. By applying this result on inequality (3) we obtain:

$$\beta(t) \geq (C \oslash \alpha^l)(t), \ \forall t \geq 0 \quad (4)$$

Note that $\beta(t)$ in inequality (4) is defined in terms of the number of stream objects that need to be processed within any time interval of length t. To obtain the equivalent service in terms of processor cycles, we can use the function $\gamma^u(k)$ defined above. The minimum service that needs to be guaranteed by the processor to ensure that the playout buffer never underflows is given by:

$$\gamma^u(\beta(t)) = \gamma^u((C \oslash \alpha^l)(t)) = \gamma^u(C(t) \oslash \phi^{u^{-1}}(rt)) \quad (5)$$

[1]The min-plus convolution operator $\otimes$ is defined as follows. For any two functions f and g, $(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$

[2]The min-plus deconvolution operator $\oslash$ is defined as follows: For any two functions f and g, $(f \oslash g)(t) = \sup_{s \geq 0}\{f(t+s) - g(s)\}$

processor cycles for all $t \geq 0$. Hence, the minimum frequency at which the processor should be run to sustain the specified playout rate is given by:

$$\min\{f \mid ft \geq \gamma^u(\beta)(t), \ \forall t \geq 0\}$$

In other words, if the processor is run at this frequency then it can be guaranteed that the playout buffer will never underflow, provided the output device starts consuming stream objects after a delay of d time units.

5. EXPERIMENTAL EVALUATION

In this section, we first explain how to generate the inputs for our framework, and then discuss how we compute the bounds formulated in Section 4. Finally, we present the results of our evaluation and analyze the tradeoffs between playout delay and processor frequency requirements for MPEG-2 and MP3 decoding.

5.1 Methodology

Simulation setup: In order to compute the minimum processor frequency using the framework described in Section 4, we need the following two inputs:

- To compute $\phi^l(k)$ and $\phi^u(k)$, we require the number of bits in each macroblock in a video stream and the number of bits in each granule belonging to an audio stream.
- We also need the processor cycle requirement for each stream object in order to compute the functions $\gamma^l(k)$ and $\gamma^u(k)$.

We obtain the number of bits present in each macroblock in a video file and the number of bits present in each granule in an audio file by analyzing the corresponding bitstreams. To compute the processor cycle requirement for each stream object we use a processor simulator to simulate the execution of each decoder task on representative audio and video clips. Towards this we used SimpleScalar [1], which is a commonly used instruction set simulator in the computer architecture domain. SimpleScalar comes with several cycle accurate simulators that differ mainly in the processor they model. We used the Sim-Profile simulator and modified the source code of the MPEG-2 and MP3 decoders with annotations that start and stop counters to track the number of processor cycles consumed by each stream object. When the modified source code is compiled using the SimpleScalar compiler, our source code annotations get transformed into user-defined assembly language instructions. We therefore extended the instruction set of SimpleScalar such that it identified our instructions for incrementing the counters during a simulation. After a simulation run for any given input file, we obtained the processor cycle requirements for every stream object in the input stream. Note that this process had to be carried out for both MPEG-2 and MP3 files independently.

Implementation issues: There are two different ways in which the proposed framework may be used. Individual MPEG-2 or MP3 clips may be profiled offline to compute the (ϕ^l, ϕ^u) and (γ^l, γ^u) functions for each clip. Using these functions the list of playout delay values and the corresponding processor cycle (or frequency) requirements can be computed (such as the list shown in Table 1). Such a list is then

Video File Characteristics	
Video Size	2 Mbyte
Number of I/B/P Frames	28/84/222
Total Number of Frames	334
Average bit rate of the video	1500 kbps
Resolution of the video files	352*240
Frame rate of the video	30fps
Length of the video	11 seconds

Table 2: MPEG-2 file characteristics.

embedded into the media file as a metadata. At runtime, the scheduler reads this metadata and delays the playout of the stream based on the current load on the processor.

A second possibility is to analyze a set of *representative* audio or video clips and compute the (ϕ^l, ϕ^u) and (γ^l, γ^u) functions for such a representative set. The assumption is that the characteristics of any new audio or video clip would be captured by these functions. Note that (ϕ^l, ϕ^u) and (γ^l, γ^u) represent lower and upper bounds on the variability in the workload. Hence, by choosing a sufficiently large representative set to derive these functions, it can be assumed that any new clip would also be bounded by them. The list of playout delay values and processor frequencies are then computed based on these functions. Clearly, this approach does not require the embedding of this list into a video or audio clip. However, for any playout delay value, the computed frequency would be much higher than the case where (ϕ^l, ϕ^u) and (γ^l, γ^u) are computed for individual clips.

We implemented the procedure for computing the minimum processor frequency corresponding to any playout delay (as explained in Section 4) using Matlab. In the next section we analyze how the minimum processor frequency varies for a number of audio and video clips. The video files we used in the experiment are `susi`, `mobile`, `cact` and `BBC` (`m2v` files obtained from [18]). Table 2 shows the characteristics of these files. Table 3 lists the audio files used (obtained from [5]) and their characteristics.

Audio File	**File Size** (in Mbytes)	**Audio Length** (sec)	**No. of Frames**
Amazing	1.86	122	4701
Christmas	2.89	182	7274
Sally	3.65	239	9189
Windows 95	1.98	130	4990

Audio File Characteristics	
Average bit rate of the audio	128 kbps
Sampling rate of the audio clips (per channel)	44.1 KHz
No. of granules in MP3 sample (constant)	576 *2

Table 3: List of MP3 files and their characteristics.

5.2 Results

Figures 3(a) and 3(b) show the playout delay versus the minimum processor frequency required to decode a set of MPEG-2 and MP3 clips. From these figures we can make the following main observations.

- For both, audio and video decoding, we see a sharp decrease in the minimum processor frequency requirement as the playout delay is increased. For example, in Figure 3(a), for a playout delay of 0.1 seconds the frequency required is in the GHz range. But as we increase the playout delay the frequency drops to the MHz range. The technique proposed in this paper exploits this drop in the processor frequency requirement to accommodate multiple tasks which could not have been run concurrently otherwise. As mentioned in Section 1, when the playout delay is very small, the minimum required processor frequency is determined by the maximum number of processor cycles required by any stream object. Since this maximum number is significantly larger than the average number of processor cycles required by any stream object, the drop in processor frequency turns out to be so sharp.

- As we continue increasing the playout delay we see that the required processor frequency stabilizes to a certain value. This value is determined by the input bitrate r (see Section 4), the average number of bits constituting each stream object (macroblock or granule) and the average number of processor cycles required to process each stream object. This frequency value may also be interpreted as the frequency with which the processor needs to be run to process a completely "smoothed out stream".

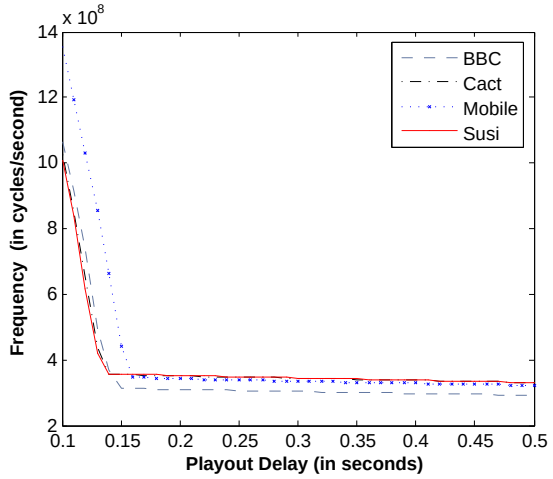
- Our framework can be used to choose a common playout delay for multiple streams (i.e. MPEG and MP3), that need to be played out in a synchronous fashion. The common playout delay is chosen such that the sum of the processor frequencies required by each stream does not exceed the maximum frequency with which the processor can be run.

- Note that in Figures 3(a) and 3(b) we plot the processor frequency values starting at 0.1 seconds. This is because the playout delay should clearly be greater then zero—it requires a certain amount of time for the first stream object to be processed and written into the playout buffer.

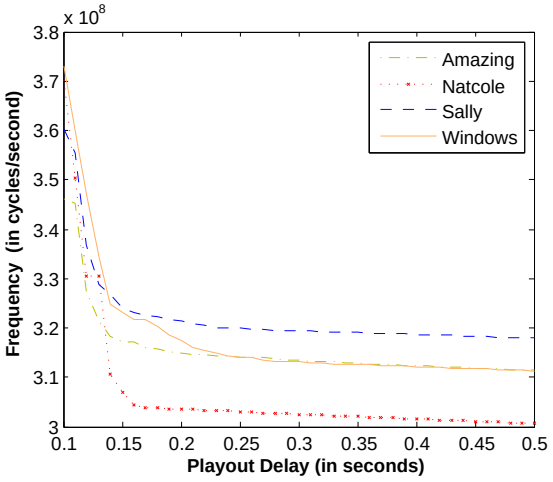
6. CONCLUDING REMARKS

In this paper we presented an analytical framework using which it is possible to compute the tradeoffs between playout delay and the minimum required processor frequency for processing multimedia streams. This is achieved by exploiting the high variability in multimedia workloads. The framework can account for both, the data-dependent variability in the processor cycle requirements of stream objects, as well as the variability in the input-output rates of multimedia streams.

Although our discussion in this paper focussed on computing the initial playout delay of a multimedia stream, the proposed framework can also be used in a dynamic environment. Suppose that a processor is currently executing an audio decoding task and the user wants to launch a video decoding application as well. Although there is some processor bandwidth available after running the audio decoder, it is not sufficient to accommodate the video decoding task. In this situation, the spare processor bandwidth can be used to "speed-up" the execution of the audio decoder such that its playout buffer fills up beyond a certain level. Thereafter it is run at a lower processor frequency and the freed processor bandwidth is allocated to the video decoding task. Our framework can be used to compute (i) the level to which the playout buffer of the audio decoder should be filled, (ii) the frequency bandwidth that is to be allocated to the audio and



(a) MPEG-2



(b) MP3

Figure 3: Processor frequency and playout delay tradeoffs.

the video decoder, and (iii) the playout delay of the video decoding task.

The proposed framework can be extended in several directions. We plan to implement our framework inside a real scheduler. We also intend to conduct rigorous experiments with voltage/frequency scalable processors and extend the framework to account for the energy consumed by a processor. Finally, note that the functions (ϕ^l, ϕ^u) and (γ^l, γ^u) capture the worst-case bounds associated with a stream (or a class of streams). Since most multimedia applications require only soft real-time guarantees, we would like to relax these bounds by ignoring worst-case scenarios that occur very rarely. This would lead to better utilization of the available processing resources, albeit at the cost of a small deterioration in the output quality.

7. REFERENCES

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[2] J.-Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, 2001.

[3] L. Cai and Y.H. Lu. Dynamic power management using data buffers. In *DATE*, Paris, France, 2004.

[4] K. Choi, R. Soma, and M. Pedram. Off-chip latency-driven dynamic voltage and frequency scaling for an MPEG decoding. In *DAC*, San Diego, California, June 2004.

[5] Digital audio systems. http://www.das.iocon.com.

[6] A. Dudani, F. Mueller, and Y. Zhu. Energy-conserving feedback EDF scheduling for embedded systems with real -time constraints. In *LCTES-SCOPES*, Berlin, Germany, June 2002.

[7] C.J. Hughes, J. Srinivasan, and S.V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *MICRO*, Austin, Texas, December 2001.

[8] C. Im and S. Ha. Dynamic voltage scaling for real-time multi-task scheduling using buffers. In *LCTES*, Washington, DC, June 2004.

[9] R. Jejurikar and R. Gupta. Dynamic volatge scaling for system wide energy minimization in real-time embedded systems. In *ISPLED*, Newport Beach, California, August 2004.

[10] Y. Liu, A. Maxianguine, S. Chakraborty, and W.T. Ooi. Processor frequency selection for SoC platforms for multimedia applications. In *RTSS*, Lisbon, Portugal, December 2004.

[11] C. Lm and S. Ha. An energy optimization technique for latency and quality constrained video applications. In *ESTIMEDIA*, Newport Beach, California, June 2003.

[12] Y. Lu, L. Benini, and G.D. Micheli. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Transacations on Computer-Aided Design of Integrated Circuits and Systems*, 21(11), November 2002.

[13] S.B. Moon, J. Kurose, and D. Towsley. Packet audio playout delay adjustment: performance bounds and algorithms. *Multimedia Systems*, 6:17–28, 1998.

[14] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive playout mechanism for packetized audio applications in wide area networks. In *INFOCOMM*, Toronto, Canada, June 1998.

[15] M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol. Robust media processing in a flexible and cost-effective network of multi-tasking coprocessors. In *14th Euromicro Conference on Real-Time Systems (ECRTS)*, 2002.

[16] D. Son, C. Yu, and H. Kim. Dynamic voltage scaling on MPEG decoding. In *ICPDS*, Kyongju City, Korea, June 2001.

[17] M. Tamai, T. Sun, K. Yasumoto, N. Shibata, and M. lto. Energy-aware video streaming with QoS control for portable computing devices. In *NOSSDAV*, Country Cork, Ireland, June 2004.

[18] Tektronix. ftp://ftp.tek.com/tv/test/streams/Element/index.html.

[19] C. Wang, J. Ho, R. Chang, and S. Hsu. A feedback-controlled EDF scheduling algorithm for real-time multimedia transmission. Technical Report TR-IIS-01-008, Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC, 2001.