Chameleon: Application Controlled Power Management with Performance Isolation

Xiaotao Liu Prashant Shenoy
xiaotaol@cs.umass.edu shenoy@cs.umass.edu

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

Abstract

In this paper, we present Chameleon—an application controlled dynamic voltage and frequency scaling approach for reducing energy consumption in mobile processors that see multimedia workloads. Our approach exports the entire responsibility of power management to the application level. Since multimedia applications impose soft real-time constraints, a key goal of our approach is to reduce energy consumption of such applications without degrading performance. We propose an operating system interface that can be used by Chameleon-aware applications to achieve energy savings and demonstrate its effectiveness for three common applications—a video decoder, a video conferencing tool, and a web browser. We implement our approach in the Linux kernel running on a Sony Transmeta laptop. Our experiments show that, compared to the traditional system-wide CPU voltage scaling approaches, our technique can achieve up to $34 - 50\%$ energy savings while delivering comparable or better performance to applications, and Chameleon is also more effective at scheduling a mix of concurrent applications with diverse energy needs.

1 Introduction

Recent technological advances have led to a proliferation of mobile devices such as laptops, personal digital assistants (PDAs), and cellular telephones with rich audio, video, and imaging capabilities. While the processing, storage, and communication capabilities of these devices have improved as predicted by Moore's law, these advances have significantly outpaced the improvements in battery capabilities. Consequently, energy continues to be a scarce resource in such devices. The situation is exacerbated by the resource-hungry nature of multimedia applications—such applications consume energy by accessing, processing, and rendering large amounts of multimedia data.

Modern mobile devices attempt to use energy judiciously by incorporating a number of power management features. For instance, modern processors such as Intel's XScale and Pentium-M and Transmeta's Crusoe incorporate dynamic voltage and frequency scaling (DVFS) capabilities. DVFS enables the CPU speed to be varied dynamically based on the workload and reduces energy consumption during periods of low utilization [12, 13, 21]. Since multimedia applications impose soft real-time constraints, voltage and frequency scaling techniques must be carefully designed to prevent the processor slowdown from interfering with the timeliness constraints of the application.

A number of hardware and software power management techniques have been developed to take advantage of DVFS-capable processors. For instance, Transmeta's LongRun is a hardware technique that measures processor utilization at the hardware level and varies the CPU speed based on the measured system-wide utilization [10]. Software approaches for DVFS have been implemented either in the operating system or at the application level. Operating system implementations of DVFS techniques determine a system-wide CPU setting based on the processor demands of the currently active tasks [8, 9, 14, 15]. In this approach, individual applications do not have any direct control over the CPU power settings. A single system-wide CPU setting is determined, typically based on the needs of the most resource-hungry application, even when a mix of applications is executing on the processor. Furthermore, the operating system needs to *infer* the processing needs of the applications using online measurements and can incur estimation errors. Application-level DVFS techniques have been studied in the communities [5, 16, 17, 19, 29] . These techniques consider a single application such as a video player and grant complete control of the processor frequency and voltage settings to the application. The power-aware application can choose a voltage and frequency

setting based on its needs and typically ignores other applications in the system. As a result, the performance of other applications can be significantly impacted when the settings chosen by the power-aware application do not satisfy their CPU needs.

A hybrid approach for DVFS was proposed in [22]. In this approach, periodic multimedia applications convey their periods and the amount of work in each period to the operating system using system calls. The technique integrates DVFS with a real-time CPU scheduling algorithm such as Earliest Deadline First (EDF) and makes scheduling and voltage scaling decisions using (i) application supplied information and (ii) probability distributions of measured application demands. Thus, while applications do not directly control their CPU power settings, they provide useful information to the OS kernel, and thereby influence these settings.

In this work, we argue that applications know best what their resource and energy needs are, and consequently, we explore an approach where applications have complete control over their CPU power settings. Unlike prior approaches, however, we use OS mechanisms to isolate applications from one another. Thus, each application can specify its CPU power settings independently of other applications, and an application is completely isolated from the settings used by other applications. Application-specific power settings enables a mix of diverse applications to flexibly optimize their energy needs. Our approach resembles the philosophy of the *Exokernel*, where the OS grants complete control of various resources to the applications and only enforces protection to prevent applications from harming one another [7].

Our current work differs from past work on DVFS in many different respects.

- Many existing OS-based DVFS approaches specifically assume periodic and/or interactive applications and do not work for other types of applications. Similarly, much of the work on application-based DVFS approaches has only considered a single application such as a video player. By exporting power management to the application-level, our approach can be used by any application, regardless of its nature. We demonstrate the benefits of our approach for periodic applications such as video players and video conferencing as well as aperiodic applications such as web browsers. We are also developing a power-aware version of an open-source Office application suite.

- Many OS-based DVFS approaches employ a single system-wide setting for all applications. Such an approach can be sub-optimal when scheduling a mix of applications, since less-resource intensive applications waste energy by using a higher setting that is necessary, and more-resource intensive applications see degraded performance when a lower setting is used than is necessary. Our approach uses a different power setting for each application, and we experimentally demonstrate its flexibility when scheduling a mix of concurrent applications with different resource needs. We note that per-process power settings are also used in [22], although the approach restricts itself to a mix of periodic applications.

- OS-based approaches can incur estimation errors since they infer the resource usage of an application by treating it as black-box. Since applications are responsible for managing their power settings in our approach, they can employ domain-specific knowledge to better infer their resource needs. For instance, a power-aware video player can employ video-specific information such as the frame size and the frame type to determine frame decoding times (and an appropriate CPU power setting). Such information is typically hard to infer within the operating system kernel. An intermediate approach is to enable an application to provide useful hints to the OS kernel (such as their periods [22]); our approach is more radical since it puts the entire burden of power management on the application.

This paper presents Chameleon, our approach for application controlled DVFS with performance isolation. Chameleon consists of three components: (i) a *common OS interface* that can be used by power-aware applications to measure their CPU demands and adjust their CPU speed settings, (ii) a modified kernel *CPU scheduler* that supports per-process CPU speed settings and ensures performance isolation among tasks (the terms applications, tasks and processes are used interchangeably in this paper), and (iii) a *speed adapter* that maps these CPU speed settings to the nearest speed actually supported by the hardware. In Chameleon, each power-aware application needs to employ a model of its resource usage. The model is parameterized by online measurements of the resource usage to determine an appropriate power setting at run-time. We present application models for three open-source applications: (i) an MPEG-2 and MPEG-4 video decoder that is representative of DVD players and commercial streaming systems, (ii) a H.261 and H.263-based video conferencing tool, and (iii) a web browser.

We have implemented Chameleon in the Linux kernel 2.4.20-9 and have evaluated its energy efficiency on a Sony Vaio laptop equipped with Transmeta's Crusoe TM5600-667 processor [20]. Our experiments compare Chameleon with three existing OS-level DVFS approaches, namely PAST [21], PEAK [13] and AVG_n [12] and with LongRun, a hardware-based DVFS approach. Our experiments with the above power-aware applications show that Chameleon can extract up to a 35% energy savings when compared to LongRun and up to 50% savings when compared to OS-based DVFS approaches, without any performance degradation to time-sensitive multimedia and interactive applications. In case of the web browser, for instance, the average power consumption of Chameleon is only 0.03W higher than the lowest power setting of Transmeta's Crusoe TM5600-667 processor. Chameleon is also more effective at scheduling a mix of applications, since each application can use a custom

power setting that is most appropriate to its needs—our experiments show that per-process power settings in Chameleon yield 31-50% energy savings over LongRun and OS approaches that use a single power setting for all applications.

The rest of this paper is organized as follows. Section 2 presents the design of Chameleon. Section 3 presents the design of Chameleon-aware applications. Section 4 discusses implementation issues. Section 5 presents our experimental results, and finally, Section 7 presents our conclusions.

2 Chameleon Design

The architecture of Chameleon consists of three key components (see Figure 1). The Chameleon *common interface* is used by power-aware applications to query the kernel for statistics on resource usage. These OS-level statistics can be combined with application domain knowledge to determine a desirable CPU power setting, which is then conveyed to the OS kernel via the common interface. Second, Chameleon implements a modified CPU scheduler that supports per-process CPU power settings and application isolation. The modified scheduler conveys an application's power settings to the underlying CPU at context switch time. Further, an application can modify its power settings at any time during its quantum via system calls. To enforce protection, an application is never allowed to modify the settings of another application. Since an application's power settings take effect *only when it is scheduled*, applications are isolated from one another and from malicious or misbehaving applications. Kernel support for per-process power settings and application isolation does not require any modifications to the CPU scheduling algorithm itself, and as a result, Chameleon is compatible with any scheduling algorithm. Third, Chameleon implements a speed adaptor that maps application-specified power settings to the nearest CPU speed actually supported by the hardware. In particular, an application specifies the desired CPU speed (and thus, its power setting) as a fraction f_i of the maximum processor speed. The speed adapter maps this fraction to the nearest supported CPU speed; since different hardware processors support different speeds, such an approach ensures portability across hardware.

While it is desirable for applications to manage their own energy needs to maximize power savings, it may not be feasible to modify every single application to make it power-aware. Thus, legacy applications will coexist with power-aware applications in Chameleon. For such applications, Chameleon reverts to a hardware DVFS technique—whenever a power-unaware application is scheduled on the CPU, Chameleon dynamically switches to a system-controlled DVFS technique (our current prototype uses LongRun [10]). The hardware DVFS technique is disabled when a power-aware application is scheduled for execution. Such a policy enables legacy applications to extract some power savings while permitting power-aware applications to maximize these savings.

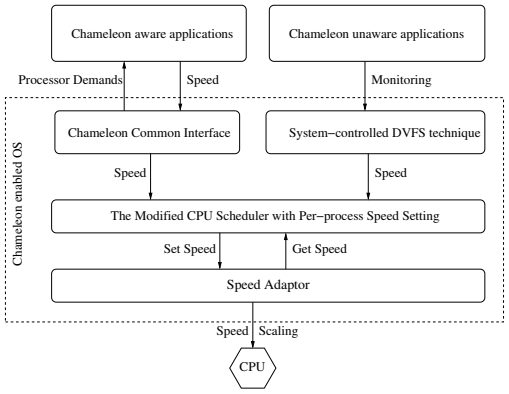


Figure 1: The Chameleon Architecture.

3 Modeling Energy Usage

Chameleon puts the burden of power management on individual applications, and consequently, each power-aware application needs a model of its resource usage to determine its power settings at run-time. Such a model predicts future resource needs and determines a power setting that is sufficient to meet those needs. In this section, we present models for three different applications, namely a video player, a video conferencing tool, and a web browser.

3.1 MPEG Video Decoder

We consider *mplayer* [18] a software video decoder that supports both MPEG-2 and MPEG-4 playback. Note that, MPEG-2 is widely used for DVD playback, while MPEG-4 is used by commercial streaming systems such as QuickTime and Windows Media; a power-aware version of *mplayer* is representative of these applications. In general, video playback is a periodic application where frames are decoded and displayed at the playback rate. The playback quality is maximized so long as each frame is decoded and displayed prior to its playback deadline. Since the decoding of a frame significantly before its playback deadline does not increase the perceived video quality, a power-aware version of a video player should vary the CPU speed so that each frame is decoded exactly when it deadline expires. To illustrate, consider a 30 frames/s video where a frame needs to be decoded every 33 ms. If the decode time of a particular frame is estimated to be 16ms at full processor speed, the speed can be effectively halved (and the decode time doubled) without impacting the 33ms deadline. More precisely, in the absence of other applications, the optimal (slowest) CPU speed f_{opt} to decode a frame is given as

$$f_{opt} = \frac{d_{max}}{\tau} f_{max} \tag{1}$$

where f_{max} is the maximum processor speed, d_{max} is the decode time of the frame at full processor speed, and τ is the playback interval. While the parameters f_{max} and τ are known for a given processor and a given video, respectively, the frame decode time d_{max} needs to be determined for each individual frame. Further, Equation 1 will need to consider the impact of time sharing due to other applications in the system.

3.1.1 Predicting Frame Decode Times

We encoded a number of MPEG-2 and MPEG-4 video clips at different bit rates and different spatial resolutions. These video clips were decoded by an instrumented *mplayer* that measured and logged the decode time of each frame at full processor speed. We analyzed the resulting traces by studying the first order and second order statistics of the decode times and frame sizes for each frame type (i.e., I, P, B) as follows.

Let x and y be two random variables corresponding to the frame size and the frame decoding time, respectively; and let μ_x and σ_x be the mean and standard deviation of the frame size, respectively; and also let μ_y and σ_y be the mean and standard deviation of the frame decoding time, respectively. Thus the theoretical correlation coefficient ρ_{xy} between x and y is given by:

$$\rho_{xy} = \frac{E[(x-\mu_x)(y-\mu_y)]}{\sigma_x \sigma_y} \tag{2}$$

Now assume we have obtained N pairs of x and y values. The correlation coefficient ρ_{xy} may be estimated from the N pairs data by:

$$r_{xy} = \frac{\Sigma_{i=1}^{N}(x_i-\overline{x})(y_i-\overline{y})}{[\Sigma_{i=1}^{N}(x_i-\overline{x})^2 \Sigma_{i=1}^{N}(y_i-\overline{y})^2]^{1/2}} \tag{3}$$

For a particular function of r_{xy} given by:

$$w = \frac{1}{2}[\frac{1+r_{xy}}{1-r_{xy}}] \tag{4}$$

From [3], the random variable w has an approximately normal distribution with a mean and variance of

$$\mu_w = \frac{1}{2}[\frac{1+\rho_{xy}}{1-\rho_{xy}}] \tag{5}$$

$$\sigma_w^2 = \frac{1}{N-3} \tag{6}$$

As shown in [3], the sampling distribution of w given $\rho_{xy} = 0$ is normal with a mean of $\mu_w = 0$ and a variance of $\sigma_w^2 = \frac{1}{N-3}$. Hence the acceptance region of the hypothesis of zero correlation at the 0.02 level of significance is given by:

$$-2.33 \leq \frac{\sqrt{N-3}}{2} ln[\frac{1+r_{xy}}{1-r_{xy}}] < 2.33 \tag{7}$$

If $\sqrt{N-3}w$ falls outside the acceptance region of zero correlation, hence, there is reason to believe that significant correlation exists between x and y.

Resolution	Frame Type	Bit-Rate(kbps)	r_{xy}	$\sqrt{N-3}w$
352x288	I	1120.0	0.8956	111.7280
352x288	P	1120.0	0.3443	47.7959
352x288	B	1120.0	0.1808	39.7774

Table 1: Correlation Coefficients of MPEG 1/2 Standard Videos

Resolution	Frame Type	Bit-Rate(kbps)	r_{xy}	$\sqrt{N-3}w$
352x240	I	630.5	0.9045	42.0324
352x240	P	630.5	0.7664	492.1438
512x288	I	705.5	0.8201	40.9122
512x288	P	705.5	0.8084	455.9816
576x256	I	775.4	0.9162	88.7301
576x256	P	775.4	0.7667	389.0298
640x272	I	1290.9	0.8824	48.3261
640x272	P	1290.9	0.6464	216.6028
640x352	I	679.7	0.6861	50.9520
640x352	P	679.7	0.8217	486.8483

Table 2: Correlation Coefficients of MPEG 4 Standard Videos

Our correlation coefficient results of the above correlation analysis in Table 1 and 2 show that there is a piece-wise linear relationship between the decode times and the frame sizes for each frame type. These results corroborate the findings of a prior study on MPEG-2 where an approximate linear relationship between frame size and decode times was observed [1].

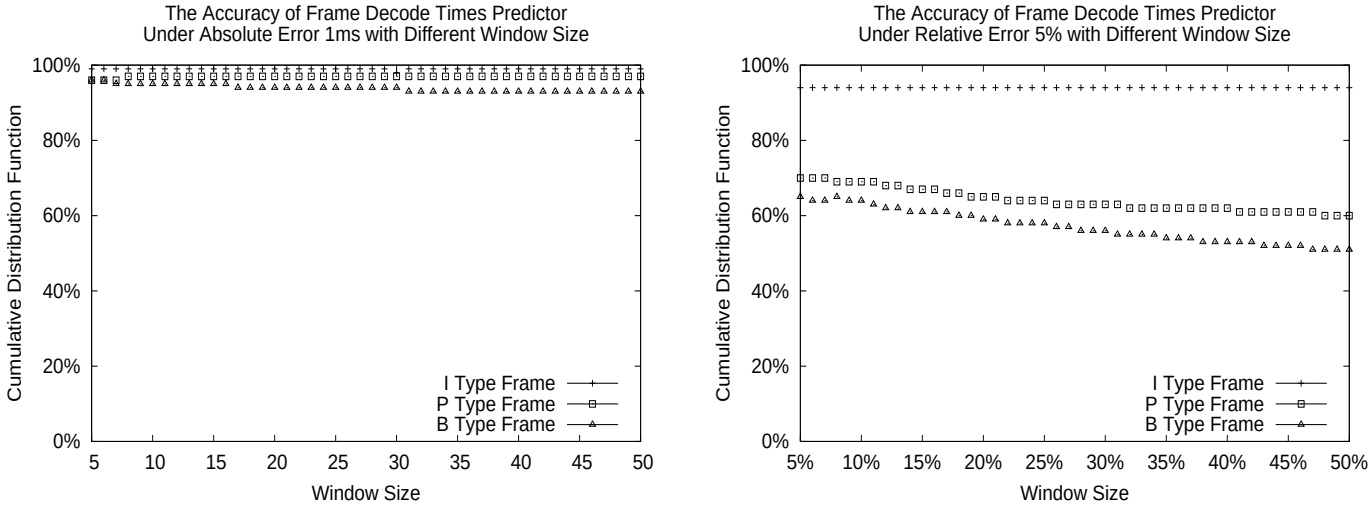
Using these insights, we constructed a predictor that uses the type and size of each frame to compute its decode time. A key feature of our predictor is that the prediction model is parameterized at run-time to determine the slope and intercept of the piece-wise linear function. To do so, the video decoder stores the observed decode times of the previous n frames, scales these values to the full-speed decode time (since the observed decode times may be at slower CPU speeds), and uses these values to periodically recompute the slopes and the intercepts of the piece-wise linear predictor by using linear regression method. This not only enables the predictor to account for differences across video clips (e.g., different bit rates require different linear predictors), it also accounts for variations within a video (e.g., slow moving scenes versus fast moving scenes in a video). The parameterized predictor is then used to estimate the decode time of each frame at full processor speed.

For instance, given window size n, suppose we have the last n I frame's size and decoding time, then we start to decode a new I frame and we already know the size of this new frame. Let s_i and d_i denote the frame size and the full-speed decoding time of the ith frame, respectively, s_{n+1} denote the frame size of the new I frame and $\hat{d}_{n+1}$ denote the predicted full-speed decoding time of it. Thus the $\hat{d}_{n+1}$ is given by Equation 8:

$$\begin{aligned} \bar{s} &= \frac{\sum_{i=1}^{n} s_i}{n} \\ \bar{d} &= \frac{\sum_{i=1}^{n} d_i}{n} \\ b &= \frac{\sum_{i=1}^{n}(s_i - \bar{s})d_i}{\sum_{i=1}^{n}(s_i - \bar{s})^2} \\ a &= \bar{d} - b\bar{s} \\ \hat{d}_{n+1} &= a + bs_{n+1} \end{aligned} \tag{8}$$

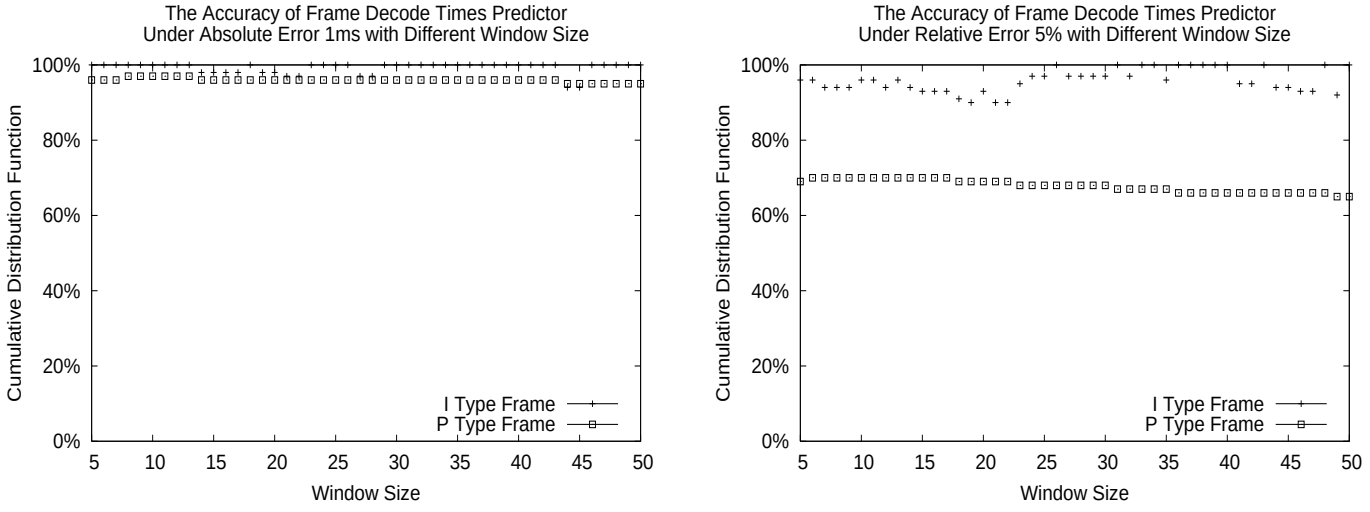
In the predictor shown in Equation 8, the window size n has great impact on the performance of the predictor, thus choosing an appropriate n is important issue in the design of such an linear regression predictor. To do this, we applied the linear regression predictor to our collected traces by varying the window size n from 5 to 50, and then measured the accuracy of the linear regression predictor with different window sizes. The accuracy of the linear regression predictor (Equation 8) is evaluated by the *Cumulative Distribution Function* (CDF) of its absolute error and the CDF of its relative error. Under the same error level, the larger the CDF, the more accurate the predictor. As shown in Figure 2 to 6, the linear regression predictor achieves the best accuracy in most cases when the window size n is less than 10, and the accuracy level has small variation in that area.

Therefore, we choose the window size 8 for our predictor since the division operations of Equation 8 can then transformed to the shift operations to reduce the cost.



(a) Absolute Error (b) Relative Error

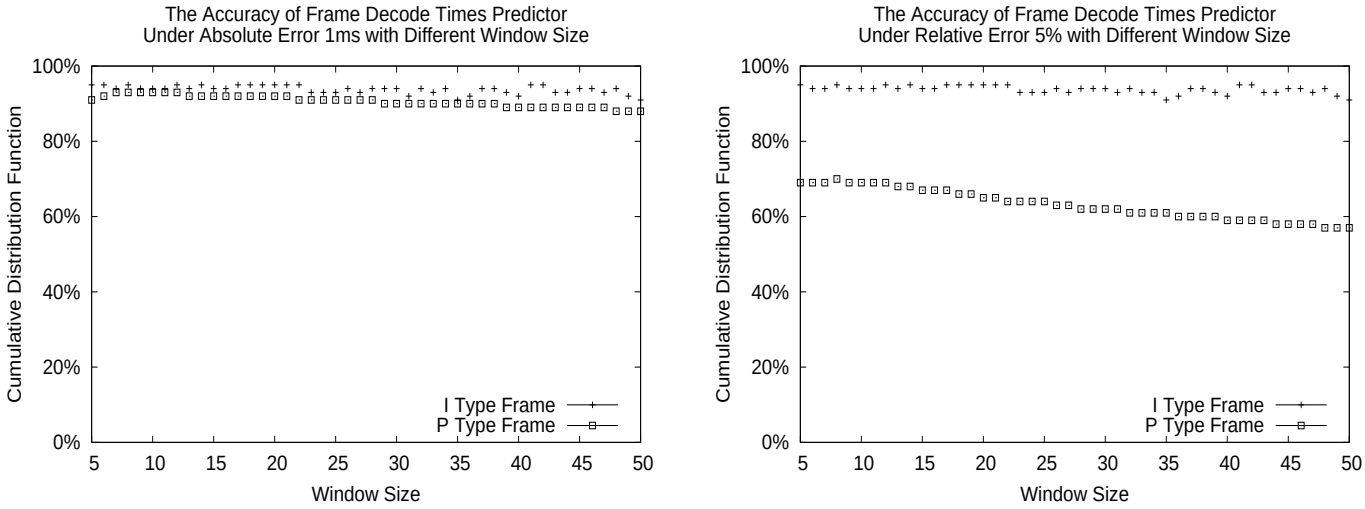
Figure 2: Variation of the Accuracy of MPEG 1/2 Frame Decode Times Predictor under Resolution 352x288 with the Window Size



(a) Absolute Error (b) Relative Error

Figure 3: Variation of the Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 352x240 with the Window Size

Figure 7, 8, 9, 10 and 11 present the accuracy of our predictors for all three different frame types (i.e, I, P, B) with window size 8. Our experiments show that our MPEG frame decode times predictor can achieve very good prediction accuracy for all frame types. Figure 7 measures the accuracy of our predictor for MPEG 1/2 movie, and Figure 8 to 11 measure the accuracy of our predictor for MPEG 4 movies. Since MPEG 4 standard only has two frame types (I and P), Figure 8 to 11 does not have the results for B type frame present. Our results show that: (i) for the decode time of I type frame, the absolute error of over 95% prediction is less than 1ms except that the absolute error of 95% prediction under resolution 640x352 is less than 2ms, and the relative error of over 92% prediction is less than 5%; (ii) for the decode time of P type frame, the absolute error of over 92% prediction is less than 1ms, and the relative error of over 90% prediction is less than 10%; (iii) for the decode time of B type



(a) Absolute Error (b) Relative Error

Figure 4: Variation of the Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 512x288 with the Window Size

frame, the absolute error of over 95% prediction is less than 1ms, and the relative error of over 88% prediction is less than 10%.

3.1.2 Speed Setting Strategy

Suppose that predicted decode time of the next frame at full processor speed is $\hat{d}_{max}$. Then the predicted optimal processor speed for decoding the next frame is given as

$$f_{predict} = \begin{cases} min(\frac{\hat{d}_{max} \cdot f_{max}}{deadline - delay}, f_{max}) & \text{if } deadline > delay \\ f_{max} & \text{if } deadline \leq delay \end{cases} \quad (9)$$

where f_{max} denotes the full processor speed, $deadline$ is the relative deadline for decoding the frame and is given by Equation 10, and $delay$ is the accumulated slack and is given by Equation 11. The deadline for decoding a frame is the actual time left until its playback instant

$$deadline = max(T - current, \epsilon) \quad (10)$$

where T denotes the playback instant, $current$ denotes the current time, and ϵ is a small positive constant. We take the maximum of $T - current$ and ϵ to avoid negative values of the deadline, in which case $f_{predict}$ should be set to f_{max}.

Since the predictor is not perfect, the actual decode time can be smaller or greater than the predicted decode time, resulting in positive or negative slack for decoding future frames. The parameter $delay$ estimates this slack (which is also the error in the predictions). The computed deadline is then reduced by this amount to correct for the error in Equation 9. The accumulated slack is computed as

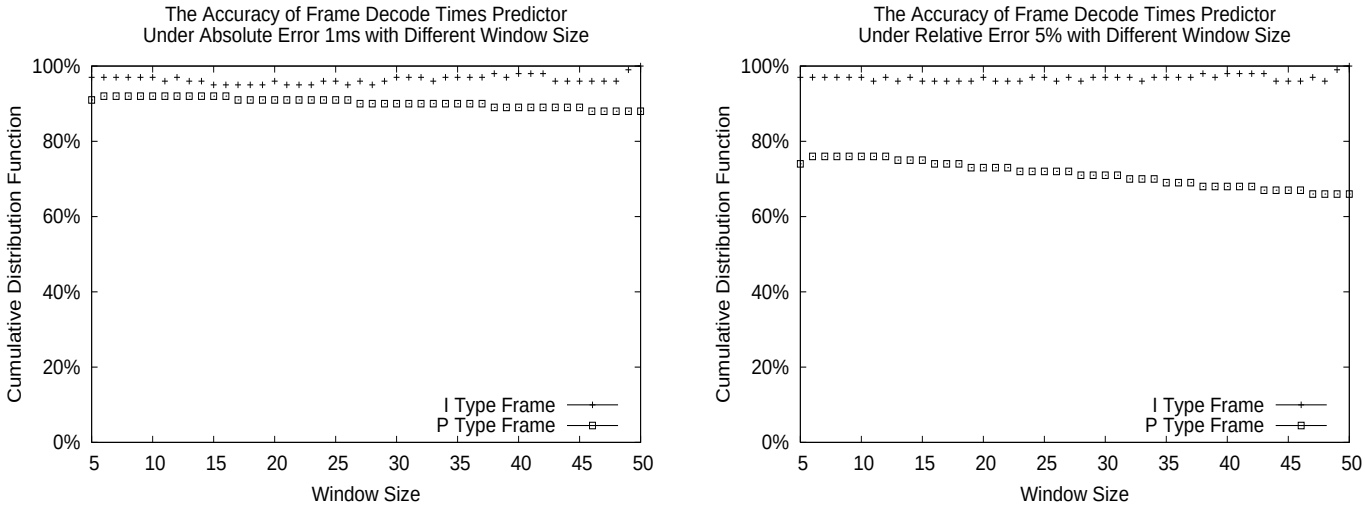
$$delay = \begin{cases} max(d_{last} - deadline_{last}, 0) & \text{if } delay \leq 0 \\ delay + (d_{last} - deadline_{last}) & \text{if } delay > 0 \end{cases} \quad (11)$$

where d_{last} and $deadline_{last}$ denote the decoding time and relative deadline of the previous frame, respectively. Note that, the use of the current time in the $deadline$ computation and the computation of the slack allows the predictor to account for the time spent on scheduling other processes in a time-shared system.

In a real implementation, the Chameleon speed adaptor maps the computed $f_{predict}$ to the closest supported CPU speed that is no less than the requested speed.

3.2 Video Conferencing Tool

Video conferencing has become a popular multimedia application for business and personal use. Most instant messaging clients today support some form of “video chats”. Business use of video conferencing has increased due to falling network bandwidth prices and better provisioned networks. Many video conferencing applications are based on the H.26x family of compression



(a) Absolute Error (b) Relative Error

Figure 5: Variation of the Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 640x272 with the Window Size

standards (specifically, H.261, H.263 and H.264). In a typical application such as *gnomemeeting*, the sender captures images at a constant rate using a capture device such as a web camera, encodes these images into frames as specified by the H.26x standard, and transmits encoded frames to a receiver over an IP network. Due to the limit on the maximum packet size in IP, frames are partitioned into packets prior to transmission; each packet can be independently decoded at the receiver without waiting for other packets of that frame, thereby reducing latency—an important factor in conferencing applications.

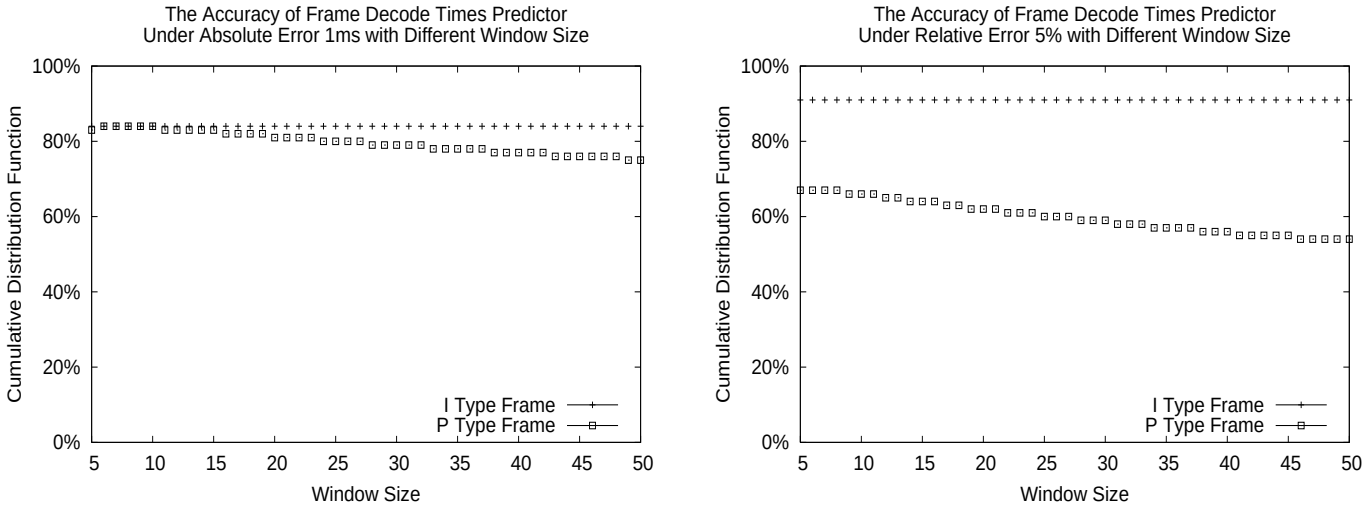
The H.26x standard differs significantly from MPEG in the techniques used for encoding and transmitting frames. In particular, although images are captured at a constant rate at the sender, the H.26x standard only requires the *difference* between successive images to be sent to the receiver. If there is no difference between successive images (due to lack of motion), then no data needs to be sent. Consequently, frames (and thus, packets) arrive *aperiodically* at the receiver, and the frame sizes can vary significantly across successive frames depending on the amount of motion. As a result, the design of a power-aware H.26x decoder is more complex than a power-aware MPEG video decoder. In particular, a power-aware video conferencing tool will need to predict (i) interval between successive frames, and specifically, the arrival time of the next frame, (ii) the size of each frame in terms of the number of packets, and (iii) the decoding time of each packet. Next, we present techniques to predict these metrics for the H.261 standard.

3.2.1 Predicting the Frame Interval

As indicated above, the interval between successive frames is not fixed in H.261, and network conditions can further add to this variability. Our experiments have shown that the amount of motion of the participants, rather than the network conditions, is the dominant factor in the variability of the frame arrival times seen by the recipient. The greater the amount of motion, the greater the difference between successive images, and the larger is the number of frames actually sent out.

To predict the arrival time of the next frame, we assume that the video conferencing application maintains a history of arrival times of the previous n frames, yielding a time series of their values. We can then use a simple time series-based statistical model to predict the next frame arrival time. We instrumented *gnomemeeting* to record the arrival times of frames and collected traces of a number of video conferencing sessions with varying amounts of motion. Using these traces, we experimented with a number of auto-regressive and moving average models such as AR(1), AR(2), AR(3), MA(1), and MA(2) to predict the next frame arrival time [2]. Except for the above time series-based statistical models, we also experimented with two commonly used models, *mean* which makes prediction by taking the mean of the values of last n samples, and *last* in which the prediction is exactly the value of last sample. Similar to Section 3.1.1, the accuracy of these predictors is also evaluated by the CDF of their absolute error and the CDF of their relative error.

As shown in Figure 12 and 13, the second-order auto-regressive model (AR(2)) and the third-order auto-regressive model (AR(3)) are the best two models, and they have similar performance. Figure 12 shows that for video conference with resolution 176x144: (i) over 97% predictions of AR(2) and AR(3) have absolute error less than 20ms; (ii) over 90% predictions of AR(2)



(a) Absolute Error (b) Relative Error

Figure 6: Variation of the Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 640x352 with the Window Size

and AR(3) have relative error less than 5%. Figure 13 shows that for video conference with resolution 352x288: (i) over 90% predictions of AR(2) and AR(3) have absolute error less than 20ms; (ii) over 95% predictions of AR(2) and AR(3) have relative error less than 10%. It means that AR(2) and AR(3) both are good candidates of frame interval predictor. Considering the computational complexity of these two predictors, AR(2) is the best choice. Consequently, we we devised a predictor based on the AR(2) model to predict frame arrival times.

To understand how the AR(2) predictor works, consider a sequence of observations of the frame intervals: $inter_0$, $inter_1$, $inter_2$, ..., $inter_n$. Given this time series, we wish to predict the $(n+1)$th frame interval. Let $inter_{n+1}$ denote the actual interval and let $\hat{inter}_{n+1}$ denote the predicted interval.

The second-order autoregressive process AR(2) is defined as [2]:

$$\tilde{inter}_t = \phi_1 \tilde{inter}_{t-1} + \phi_2 \tilde{inter}_{t-2} + a_t \tag{12}$$

where a_t is some random variable with zero mean, $\phi_1 + \phi_2 < 1$, $\phi_2 - \phi_1 < 1$, and $-1 < \phi_2 < 1$. If $inter_t$ has a non-zero mean μ, then $\tilde{inter}_t = inter_t - \mu$, otherwise, $\tilde{inter}_t = inter_t$.

Given such a process, an *AR(2) predictor* estimates the mean of $inter_t$, the parameter ϕ_1 and ϕ_2 of the model and then predicts the next value based on these estimates. Let $\hat{\mu}$, $\hat{\phi}_1$ and $\hat{\phi}_2$ denote the estimated mean, the estimated value of ϕ_1 and ϕ_2, respectively. The prediction $\hat{inter}_{n+1}$ is given by:

$$\hat{inter}_{n+1} = \hat{\mu} + \hat{\phi}_1(inter_n - \hat{\mu}) + \hat{\phi}_2(inter_{n-1} - \hat{\mu}) \tag{13}$$

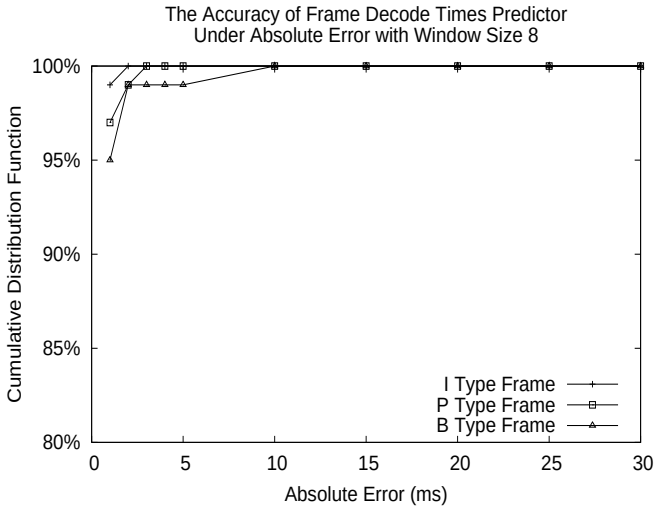
Thus, estimation of the mean $\hat{\mu}$, the parameter $\hat{\phi}_1$, and the parameter $\hat{\phi}_2$ are important issues in the design of an AR(2) predictor.

Our predictor estimates these three parameters dynamically using recent observations. Consider a window that hold the most recent m observations of frame intervals, $m \leq n$. The estimate of the mean $\hat{\mu}$ is given by:

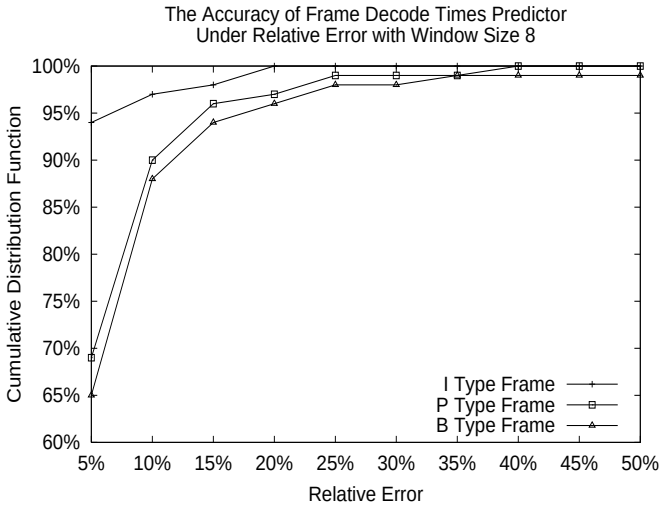
$$\hat{\mu} = \frac{\sum_{j=0}^{m-1} inter_{n-j}}{m} \tag{14}$$

The estimate of $\hat{\phi}_1$ and $\hat{\phi}_2$ are given by:

$$\begin{aligned} \hat{\phi}_1 &= \frac{r_1(1-r_2)}{1-r_1^2} \\ \hat{\phi}_2 &= \frac{r_2 - r_1^2}{1-r_1^2} \end{aligned} \tag{15}$$



(a) CDF of Absolute Error



(b) CDF of Relative Error

Figure 7: The Accuracy of MPEG 1/2 Frame Decode Times Predictor under Resolution 352x288

where

$$
\begin{aligned}
r_1 &= \frac{\sum_{j=0}^{m-1}(inter_{n-j}-\hat{\mu})(inter_{n-1-j}-\hat{\mu})}{\sum_{j=0}^{m-1}(inter_{n-j}-\hat{\mu})^2} \\
r_2 &= \frac{\sum_{j=0}^{m-1}(inter_{n-j}-\hat{\mu})(inter_{n-2-j}-\hat{\mu})}{\sum_{j=0}^{m-1}(inter_{n-j}-\hat{\mu})^2}
\end{aligned}
\tag{16}
$$

3.2.2 Predicting the Number of Packets in a Frame

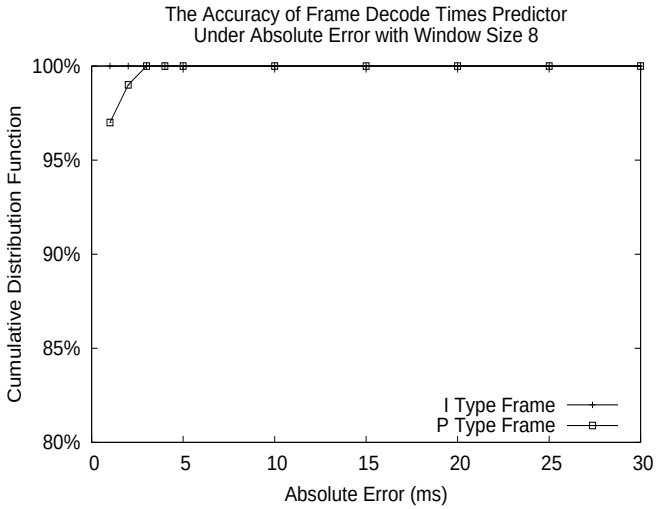
Our analysis of the gnomemeeting video conferencing traces showed that the number of packets in a frame, and thus the frame size, is governed by the amount of human motion in each frame. Due to the continuous nature of human motion, we found the size of the current frame to be the best predictor of the size of the next frame (the current frame size was found to be a better predictor than other metrics such as the mean size of the previous k frames). As shown in Figure 14 and 15, the *last* predictor which uses the number of packets in the current frame as the the number of packets in the next frame yields a good balance between prediction accuracy and computational complexity: (i) the absolute error of over 95% prediction for QCIF is less than 1; (ii) the absolute error of over 90% prediction for QCIF is less than 2. Consequently, we use a simple predictor that sets the estimated number of packets in the next frame to that in the current frame.

3.2.3 Predicting the Packet Decode Time

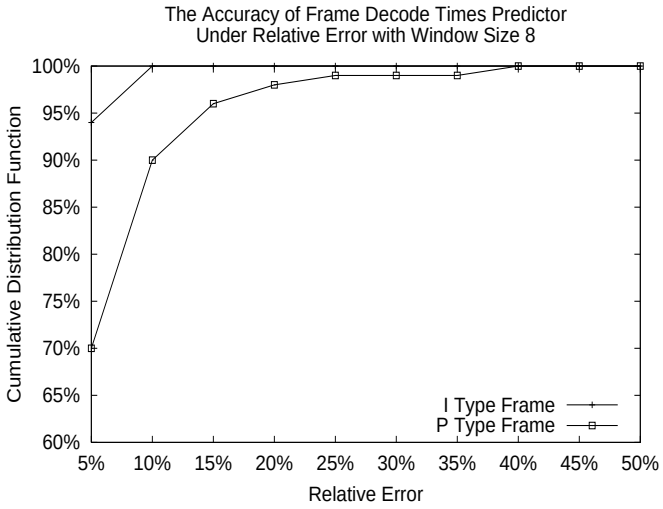
The low level compression mechanisms in H.261 share many common ideas with MPEG. Therefore, we applied the same correlation coefficients analysis as we did in Section 3.1.1 the collected traces of the packet size and the packet decoding time of H.261. Not surprisingly, we observed a similar piece-wise linear relationship between the packet size and the packet decoding time for a given frame type (see Table 3). Consequently, we use a similar predictor to the one in Section 3.1.1 to estimate the decoding time of a packet, and the equations of this predictor is exactly the equations used in Equation 8 except that the size of frames is replaced by the size of packets.

Resolution	r_{xy}	$\sqrt{N-3}w$
176x144	0.3774	36.6280
352x288	0.3176	46.4444

Table 3: Correlation Coefficients of H.261 Standard



(a) CDF of Absolute Error



(b) CDF of Relative Error

Figure 8: The Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 352x240

As we did in Section 3.1.1, we also applied our linear-based predictor (denoted as *linear*) to the traces of H.261 packets decoding time and measured its accuracy by evaluating the CDF of its absolute error and relative error. Our experiments show that our linear-based predictor of H.261 packet decoding time cannot achieve comparable accuracy to the linear-based predictor of MPEG frame decoding time in terms of the CDF of relative error. Therefore, we additionally applied another predictor *mean*, which makes prediction by taking the mean of the values of last n samples, to our collected traces. Figure 16 and 17 show that: (i) our linear-based predictor *linear* outperforms *mean* predictor in both resolutions; (ii) more than 95% predictions of *linear* have absolute error less than 1.5ms.

We notice that both predictors do not perform well in terms of the CDF of relative error. Observed from Figure 18, the decoding time of H.261 standard is small in most case, around 90% less than $3.0ms$. So even absolute error $1.5ms$ means a relative error larger than 50% in most case, while a prediction with $1.5ms$ absolute error is a very good prediction to satisfy our requirement. Therefore, we only evaluated the prediction accuracy of both predictors in terms of the CDF of absolute error. As a consequence, we choose our linear-based predictor *linear* as the H.261 packet decoding time predictor.

3.2.4 Speed Setting Strategy

Let $\hat{p}$ denote the estimated number of packets in the current frame, and let $\hat{inter}$ denote the predicted frame interval. Then the CPU speed f_j for decoding of the jth packet in the current frame is determined by scaling its full-speed decode time $\hat{d}_j$ by the inter-packet arrival time. That is,

$$f_j = \begin{cases} min(f_{max}, \frac{\hat{d}_j \cdot f_{max}}{\frac{j \times \hat{inter}}{\hat{p}} - elapse_j}) & \text{if } \frac{j \times \hat{inter}}{\hat{p}} > elapse_j \\ f_{max} & \text{if } \frac{j \times \hat{inter}}{\hat{p}} \leq elapse_j \end{cases} \tag{17}$$

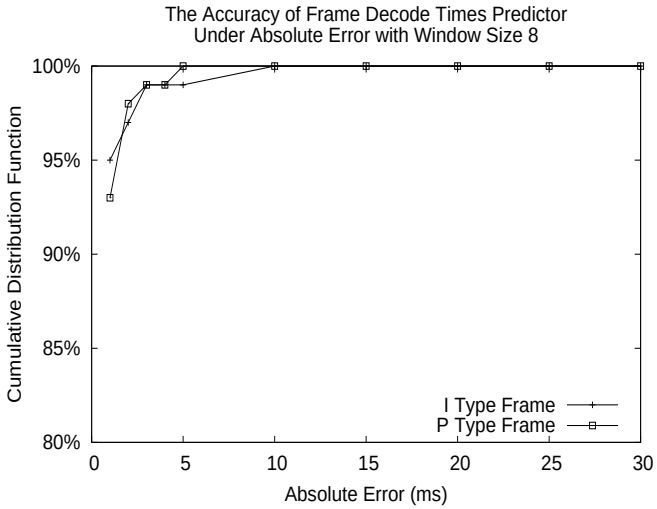
where f_{max} denotes maximum CPU speed, and $elapse_j$ denotes the time elapsed since the arrival of the first packet of the current frame.

In the event the actual number of packets in the frame exceeds the estimated value $\hat{p}$, the CPU speed setting of subsequent packets is computed as

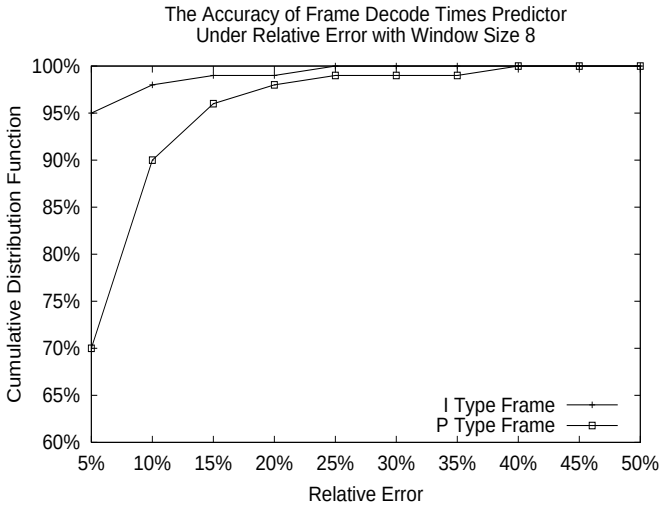
$$f_j = \begin{cases} min(f_{max}, \frac{\hat{d}_j \cdot f_{max}}{\hat{inter} - elapse_j}) & \text{if } elapse_j < \hat{inter} \\ f_{max} & \text{if } elapse_j \geq \hat{inter} \end{cases} \tag{18}$$

where $j > \hat{p}$.

In a real implementation, the computed f_j is mapped by the speed adapter to the closest available speed that is no smaller than the requested speed.



(a) CDF of Absolute Error



(b) CDF of Relative Error

Figure 9: The Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 512x288

3.3 Web Browser

A web browser is an event-driven interactive application. Upon an event such as a mouse click or network data arrival, the web browser needs to do some work to process the event. For example, when the user clicks on a link, the browser needs to construct and send out a HTTP request; when data arrives from the remote server, it needs to parse and display the incoming data; and it needs to redraw its window once the *draw* event arrives. Except for the network delay which is out of the control of the web browser, the speed at which these events are processed by web browser greatly impacts the user's experience. The faster the speed, the better the user's experience. However, studies have shown that there exists a human perception threshold under which events appear to happen instantaneously [4]. Thus, completing these events any faster would not have any perceptible impact on the user. While the exact value of the perception threshold is dependent on the user and the type of task being accomplished, a value of 50ms is commonly used [4, 15, 8, 9]. We also use this perception threshold in our work.

Our strategy for achieving energy savings while still maintaining good interactive performance relies on a technique referred to as *gradual processor acceleration (GPA)*. Our gradual processor acceleration technique works as follows.

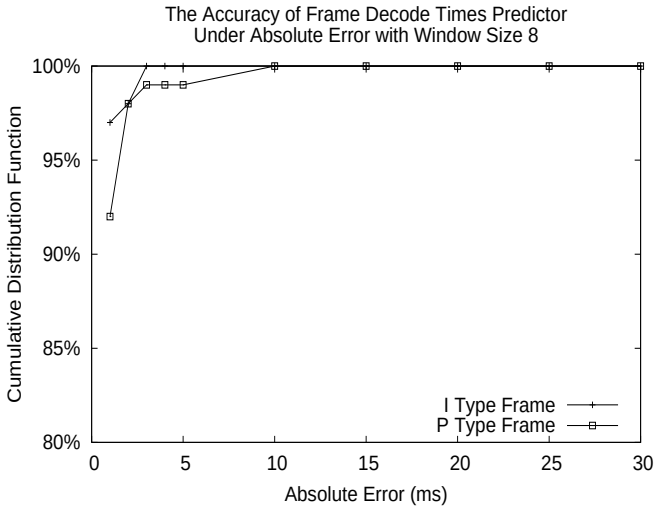
On the arrival of an event, the web browser is configured to run under at a low CPU frequency, and a timer is set. If the processing of the event finishes before the timer expires, then browser simply waits for the next event. Otherwise, it increases the CPU speed by some amount and sets another timer. Thus, the processor is gradually accelerated until either the event is processed or the maximum CPU speed is reached. In order to ensure good interactive performance, the maximum CPU speed is always used when the event processing time exceeds the perception threshold.

Suppose we have n timers, which have values t_1, t_2, ..., t_n, and $\sum_{i=1}^{n} t_i = 50$ms. At the ith step, the processor runs at speed f_i, which is expressed as a percentage of the maximum available speed. Therefore, the full speed execution time over the interval $[t_1, t_n]$—the time it would have taken to process this work at full processor speed—is given as $\sum_{i=1}^{n} f_i t_i$. If the actual full-speed processing time of the event is smaller than this value, the event finishes before the 50ms perception threshold, and thus the user does not perceive any performance degradation. For any event requiring more than this amount of full speed execution time, the maximum possible performance degradation under our strategy is given by:

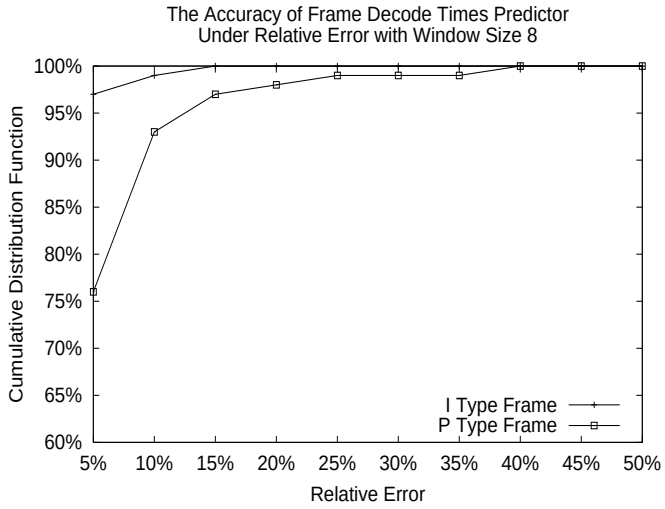
$$degrade = 50 - \sum_{i=1}^{n} f_i t_i \tag{19}$$

since the processor will run at full speed once the execution time exceeds the perception threshold.

Given this expression, the maximum possible performance degradation can be bound by any specific value by carefully choosing the CPU frequencies and timer values. For example, suppose that we have five timers with values 30ms, 5ms, 5ms, 5ms, and 5ms. Suppose the processor speeds during these timer intervals is is 45%, 60%, 80%, 90%, and 100% of the maximum speed, from the first timer to the last timer respectively. Then, from Equation 19, the maximum possible performance degradation for an event is 20ms. This is the maximum user-perceived slowdown for any event that requires more than 50ms of processing



(a) CDF of Absolute Error



(b) CDF of Relative Error

Figure 10: The Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 640x272

time.

4 Implementation of Chameleon

We have implemented Chameleon in the Linux kernel and have implemented three power-aware applications to demonstrate its effectiveness. Our implementation of Chameleon runs on a Sony Vaio PCG-V1CPK laptop with Transmeta Crusoe TM5600-667 processor [20]. The Transmeta TM5600 processor supports five discrete frequency and voltage levels (see Table 4) and implements the *LongRun* [10] technology in hardware to dynamically vary the CPU frequency based on the observed system-wide CPU utilization. LongRun varies the CPU frequency between a user-specified maximum and minimum values—these values can be set by users by writing to two machine special registers (MSR). By default, these values are set to 300 MHZ and 677 MHz, enabling the full range of voltage scaling. LongRun can be disabled by setting the minimum and maximum register values to the same frequency (e.g., setting both to 533 MHz does not allow any leeway in changing the CPU frequency, effectively disabling LongRun). This feature can be used to implement voltage scaling in *software*—the power-aware application can determine the desired frequency and set the two registers to this value.

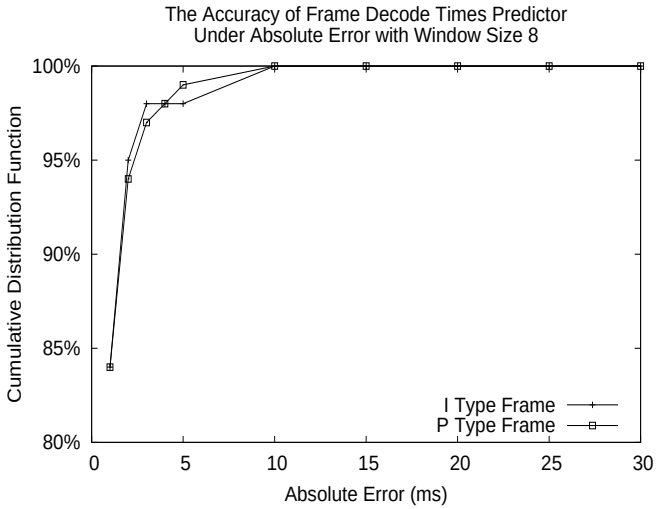
Freq. (MHz)	Voltage (V)	Power (W)
300	1.2	1.30
400	1.225	1.90
533	1.35	3.00
600	1.5	4.20
667	1.6	5.30

Table 4: Characteristics of the TM5600-667 processor

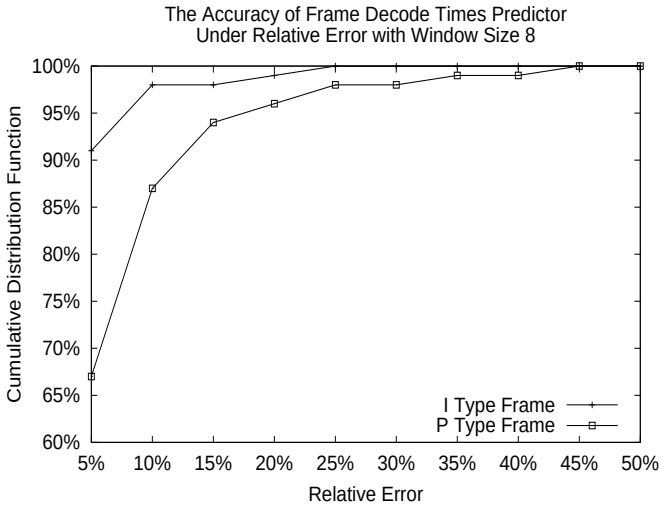
Our prototype of Chameleon is implemented as a set of modules and patches in the Linux kernel 2.4.20-9. Our prototype includes the following components:

1. **New system calls**. We added two new system calls to implement the Chameleon common interface: (i) *get-speed* which returns the current CPU speed, (ii) *set-speed* that sets the CPU speed of the calling process. We also modified the $/proc$ interface in Linux to report the full speed execution time and the per-process utilization in each quantum.

2. **Per-Process CPU Speed Settings**. We modified the Linux CPU scheduler to support per-process CPU speed settings. The scheduler maintains the current CPU speed settings for each active process and conveys these settings to the CPU at context switch time. Protection is enforced by allowing a process to only modify its own power settings and never those of other processes.



(a) CDF of Absolute Error



(b) CDF of Relative Error

Figure 11: The Accuracy of MPEG 4 Frame Decode Times Predictor under Resolution 640x352

3. **Speed Adapter**. We derived a hardware-specific conversion table (see Table 5) from off-line empirical experiments to map CPU speed percentages to a corresponding CPU frequency.

CPU Speed Percentage	Freq. (MHz)
$[0\%, 45\%)$	300
$[45\%, 60\%)$	400
$[60\%, 80\%]$	533
$(80\%, 90\%]$	600
$(90\%, 100\%]$	667

Table 5: Speed adapter mappings from the percentage CPU Speed to a CPU Frequency for the Transmeta TM5600.

4. **Power-aware Applications:** We extended the *mplayer* [18] movie player, the *gnomemeeting* video conference suite [11], and the *dillo* web browser [6] with the models presented in Sections 3.1, 3.2, and 3.3, respectively. Power-unaware applications are handled by dynamically reverting to LongRun whenever such applications are scheduled; LongRun is disabled whenever a power-aware application is scheduled.

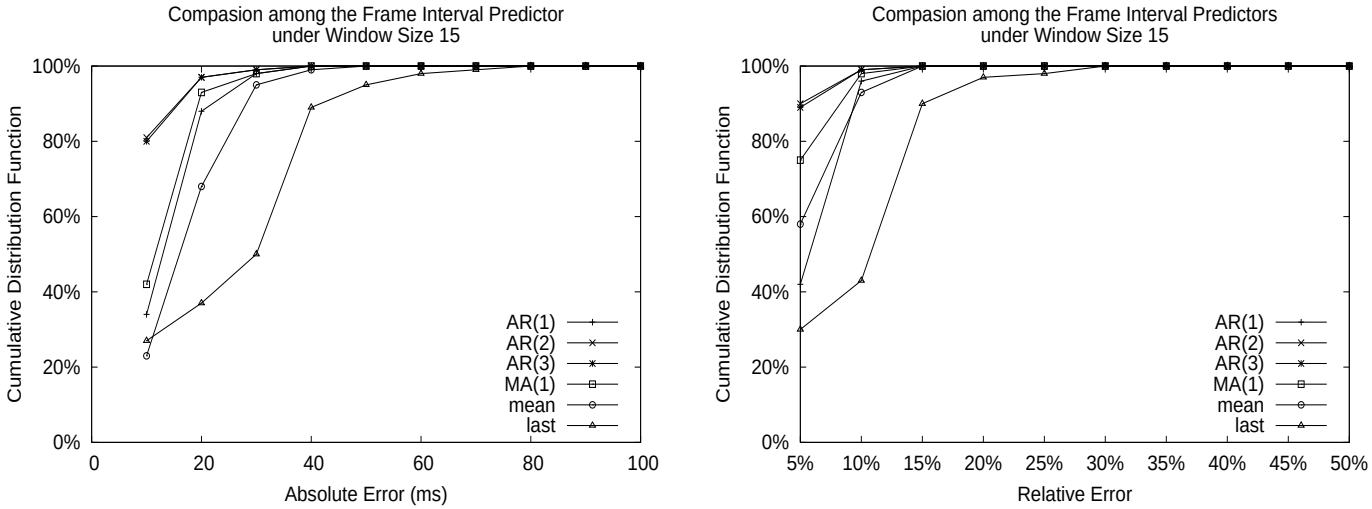
5 Experimental Evaluation

We evaluated Chameleon on a Sony Vaio PCG-V1CPK laptop equipped with a Transmeta Crusoe processor and 128MB RAM. The operating system is Red Hat Linux 9.0 with a modified version of Linux kernel 2.4.20-9. This section presents a summary of our results.

To compare Chameleon with other DVFS approaches, we implemented three OS-based DVFS techniques proposed in the literature: (i) PAST [21], (ii) PEAK [13], and (iii) AVG_n [12], all of which are interval-based system-wide DVFS techniques. Our experiments involve running applications under six different configurations: (i) with DVFS disabled—the CPU always runs at the maximum speed (denoted as FULL), (ii) using the hardwired LongRun technology, (iii) using PAST, (iv) using PEAK, (v) using AVG_n, and (vi) using Chameleon (where LongRun is disabled for power-aware applications but enabled for legacy applications).

The energy consumption of the processor during an interval T is computed as

$$energy = \sum_{i=1}^{n} p_i t_i \tag{20}$$



(a) CDF of Absolute Error

(b) CDF of Relative Error

Figure 12: The Comparisons among Frame Interval Predictors for QCIF 176x144 with Window Size 15

where n is the number of available frequency/voltage combinations on the processor, p_i denotes the power consumption of the processor when running at the ith frequency/voltage combination, and t_i represents the time spent at the ith frequency/voltage combination during the interval T. We modify the Linux kernel to record the energy consumption of the TM5600 processor using Equation 20 and Table 4. Given the energy consumption of the processor during an interval T, the average power consumption of the processor during this interval is computed as

$$power_{avg} = \frac{energy}{T} \tag{21}$$

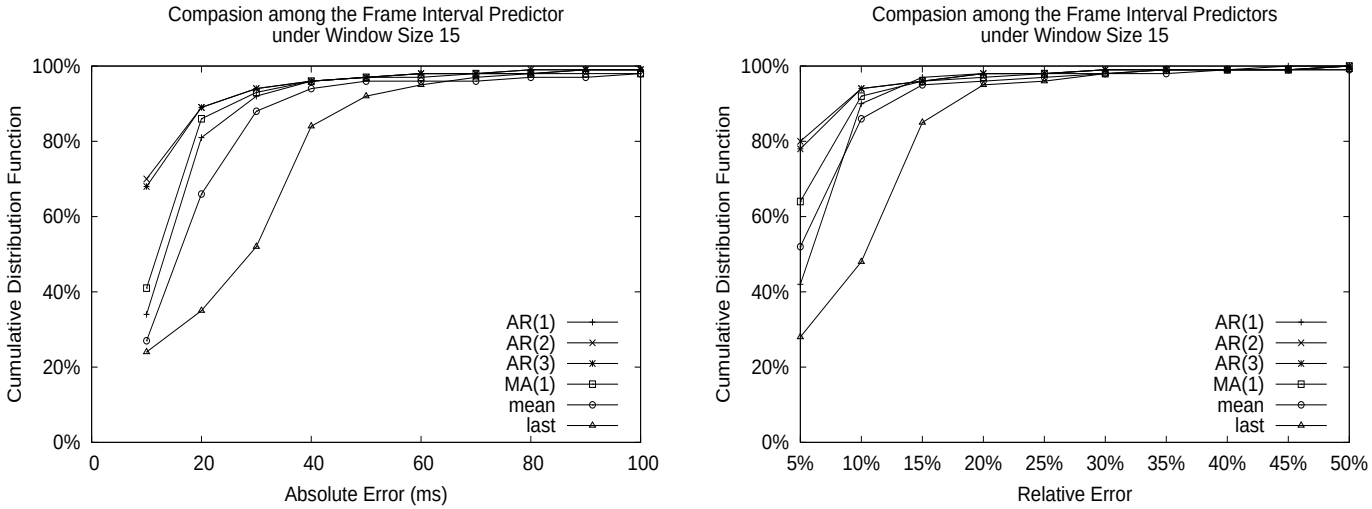
5.1 Video Decoder

We encoded ten DVD movies at different bit-rates and resolutions using Divx MPEG2/MPEG4 video codec and MP3 audio codec The characteristics of these movies are listed in Table 6. The bit-rates are depicted in the form $(a + b)$Kbps, where a is the video and b is the audio bit-rate. We recorded the energy consumed by the processor during playback of these movies at full speed, with LongRun, with Chameleon, with PAST, with PEAK, and with AVG_n.

	Res.	Length	Frames	Bit-Rate(Kbps)
Movie 1	640x272	3360s	80387	1290.9 + 179.2
Movie 2	640x272	612s	14577	757.2 + 128.0
Movie 3	720x448	1742s	43500	1272.1 + 96.0
Movie 4	640x352	602s	15003	861.9 + 128.0
Movie 5	640x352	1755s	42040	2456.9 + 192.0
Movie 6	640x480	2394s	57355	1674.6 + 384.0
Movie 7	640x352	7168s	179168	679.7 + 128.0
Movie 8	640x480	2368s	56733	1877.6 + 384.0
Movie 9	640x280	5523s	132375	911.1 + 128.0
Movie 10	720x448	1722s	43004	1250.6 + 96.0

Table 6: Characteristics of MPEG 4 Videos

As shown in Figure 19, PEAK always consumes the least processor energy among all the DVFS techniques. However, it trades its energy savings with an unacceptably high performance degradation for time-sensitive multimedia and interactive applications. For example, the results of the normalized execution time (normalized according to the length of movies) during videos playback in Figure 20 show that the video decodings of six movies (Movie 3, 4, 5, 6, 8, 10) take extra 12% to 53% execution time, resulting in poor performance. Therefore, we omit PEAK in the rest of experimental evaluation.



(a) CDF of Absolute Error

(b) CDF of Relative Error

Figure 13: The Comparisons among Frame Interval Predictors for CIF 352x288 with Window Size 15

Our results in Figure 20 show that all the other five configurations, Chameleon, LongRun, PAST, AVG_n, and Full, handle movie playback very well. The same playback quality is observed under these five configurations: identical execution times which equal the length of the movies, identical frame rates, no dropped frames, and no user-noticeable delays. However, the average CPU power consumption differs significantly across the various configurations (see Figure 19). Figure 19 shows that: (i) LongRun outperforms PAST and AVG_n in most cases; (ii) LongRun can achieve significant energy savings (from 11.76% to 62.19%) when compared to FULL; (iii) the Chameleon-aware *mplayer* can achieve an additional 19.92% to 34.79% energy savings when compared to LongRun.

Although there are no user-perceived playback problems (in terms of dropped frames or playback freezes) under the five configurations, we do observe variations in playback quality at the frame-level. *mplayer* provides statistical measurements of late frames—the number of frames that are behind their deadline by more than 20% of the frame interval (this small inter-frame jitter is typically not perceptible at the user-level). As shown in Figure 21, the number of late frames in Chameleon is mostly comparable to PAST and AVG_n and typically better than LongRun (while consuming the least energy). FULL has the least—although not zero—late frames at the expense of the the highest energy consumption. The number of late frames is small ($0.2 - 5.3\%$) in all the five configurations. We also notice that the difference between the late frames percentages of Chameleon and Full is less than 0.5% in all cases, therefore Chameleon has almost the same performance as FULL while consuming much less energy (41.71% to 71.46% less).

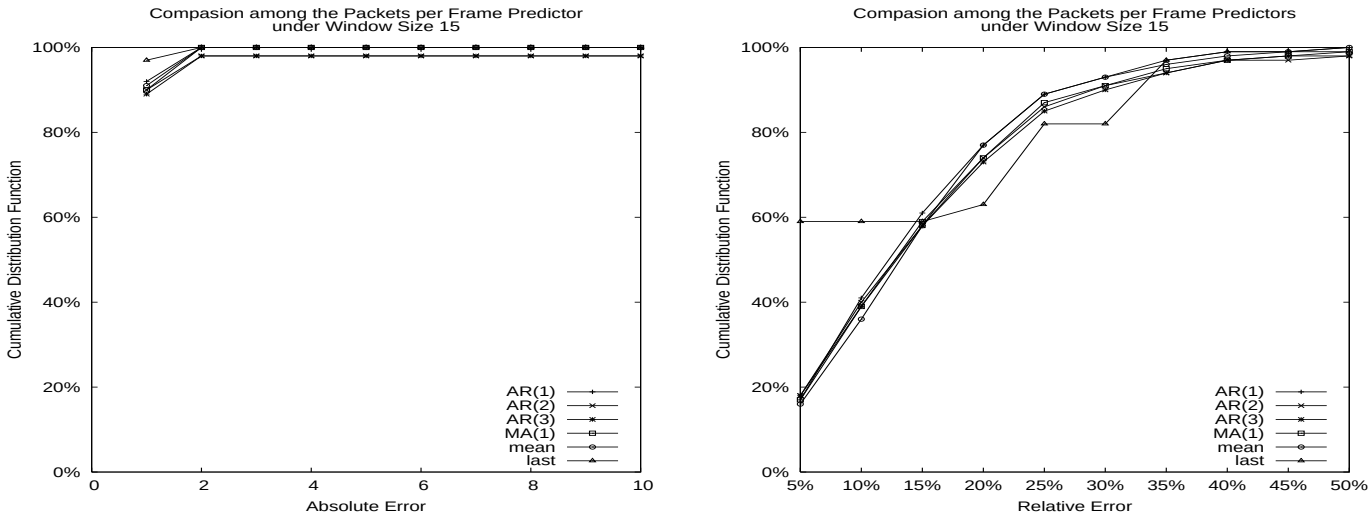
5.2 Video Conference Tool

To ensure repeatable and comparable experiments with the video conferencing tool, we encoded several video clips with varying degrees of motion. We played these clips on a PC, with the video camera of the sender PC pointing to this video playback. The sender encodes these images and transmits them to the power-aware receiver over a lightly loaded network. This ensures a fair comparison across the various DVFS techniques and enables us to carefully control the amount of motion in each session.

We ran our video conference experiments under two resolutions, QCIF (176x144) and CIF (352x288), for all five configurations. In our experiments, all five configuration handle the video conference very well. The same quality is observed under all configurations: identical execution times and no deadline misses (i.e., the decoding of each packet completes before the arrival of the next packet). Our results, shown in Figure 22, show that LongRun achieves significant energy savings (from 20.75% to 69.25%) when compared to FULL. Chameleon-aware *gnomemeeting* achieves an additional $11 - 34\%$ energy savings when compared to LongRun, while PAST and AVG_n are worse than LongRun.

5.3 Web Browser

To eliminate the impact of variable network delays, our experiments with the web browser consisted of a client requesting a sequence of web pages from a web server over a local area network; the requested web pages consist of actual web content that



(a) CDF of Absolute Error (b) CDF of Relative Error

Figure 14: The Accuracy of H.261 Packets per Frame Predictor under Resolution 176x144

was saved from a variety of popular web sites. Each experiment consists of a sequence of requests to these web pages with a uniformly distributed "think-time" between successive requests. The experiments differ in the requested web pages and the chosen think times; each experiment is repeated under the five configurations, and we measure the mean power consumption for each experiment. Our results, depicted in Table 7, show that LongRun consumes a factor of three less power than FULL. Chameleon-aware *dillo* and AVG_n are able to extract additional 10.27% and 5.41% energy savings when compared to LongRun, respectively, while PAST are worse than LongRun. We also note that the average power consumption under Chameleon is only 0.03W higher than the power consumption at the slowest CPU speed (300MHz). Further, most events finish in Chameleon without any performance degradation, and a few long events are slowed down by at most 20ms.

	Chameleon	LongRun	PAST	AVG_n	FULL
AVG. Power	1.33W	1.48W	1.88W	1.40W	5.30W

Table 7: Average Power Consumption for Web Browsing.

5.4 Concurrent Applications

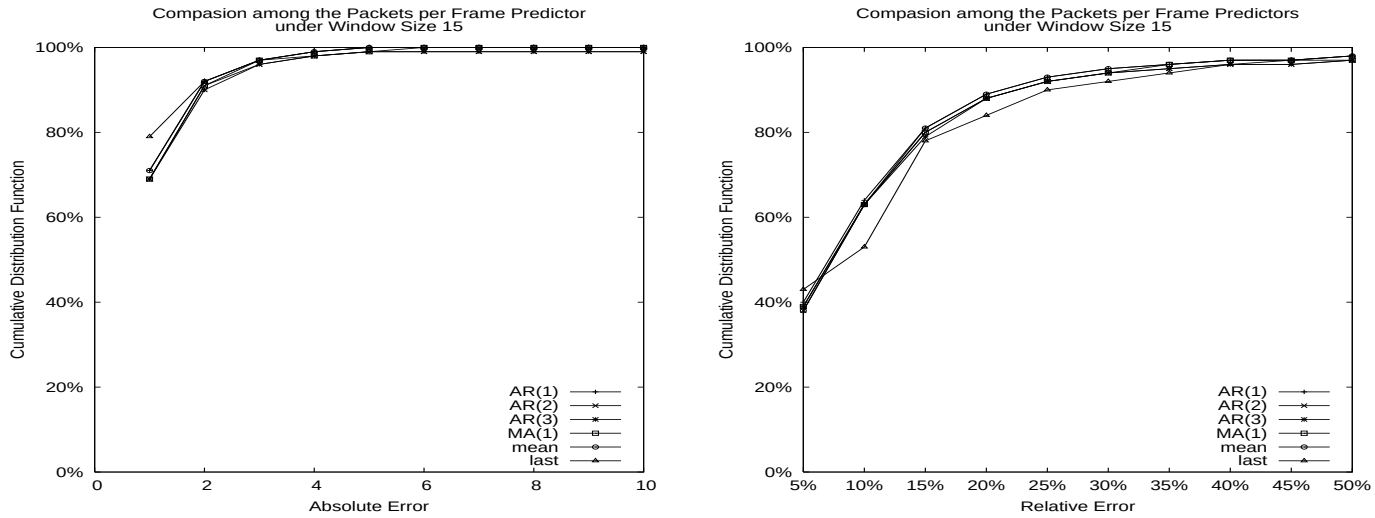
In Section 1, we hypothesized that the use of per-process power settings is better than a single system-wide setting, since it allows each application to flexibly optimize their energy consumption. To experimentally verify this hypothesis, we measure the power usage when running concurrent applications under the five configurations. We first run the video decoder and the web browser concurrently and then the video conferencing tool with the browser.

Table 8 shows the average CPU power consumption when running the video decoder and the web browser concurrently. Note that, LongRun, PAST and AVG_n determine a single system-wide setting, which is typically influenced by the most compute-intensive application in the system (in this case, the video decoder). Chameleon uses different application-computed settings for the two applications, and consequently, incurs the least power consumption of the five configurations. The energy savings range from 25-31% when compared to LongRun, which itself extracts a factor of 1.6-2 reduction when compared to FULL.

	Chameleon	LongRun	PAST	AVG_n	FULL
Movie 2	1.75W	2.33W	3.32W	3.52W	5.3W
Movie 4	2.25W	3.27W	3.98W	4.42W	5.3W

Table 8: Average CPU Power Consumption during Video Decoding and Web Browsing.

Table 9 shows the average CPU power consumption when running the video conferencing tool and the web browser concurrently. Like in the previous scenario, Chameleon is able to extract the maximum energy savings, which is 15-31% lower than



(a) CDF of Absolute Error (b) CDF of Relative Error

Figure 15: The Accuracy of H.261 Packets per Frame Predictor under Resolution 352x288

	QCIF					CIF				
	Chameleon	LongRun	PAST	AVG_n	FULL	Chameleon	LongRun	PAST	AVG_n	FULL
Conference 1	1.51W	1.79W	2.58W	2.67W	5.30W	2.91W	4.39W	4.31W	4.53W	5.30W
Conference 2	1.53W	1.80W	2.60W	2.71W	5.30W	2.93W	4.31W	4.24W	4.55W	5.30W

Table 9: Average CPU Power Consumption during Video Conferencing and Web Browsing.

than LongRun. PAST yields worse performance than LongRun; AVG_n yields worse performance than LongRun under QCIF, while it yields at most 1.82% better performance than LongRun under CIF.

5.5 Overhead

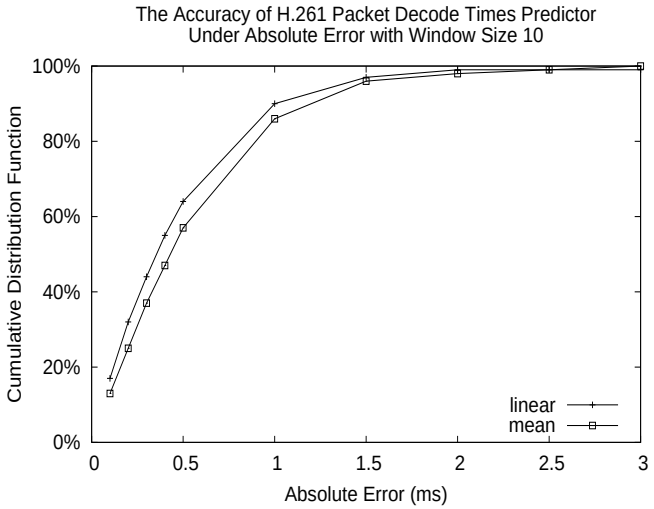
We evaluate Chameleon's overhead by measuring the cost for prediction methods and DVFS. We measure cost in CPU cycles, rather than time, since the elapsed time for an operation (e.g., an invocation of frame decode time predictor) depends on the speed, while the number of consumed cycles does not change substantially with the speed. We get the number of CPU cycles by reading the special time-stamp register of the processor before and after these operations, prediction methods and DVFS, and count the elapsed CPU cycles during them.

First, we evaluate the cost for the prediction methods in Table 10. To do this, we ran the prediction methods proposed in Section 3, and measure the elapsed CPU cycles for each prediction method. Table 10 shows that these predictor methods take less than 2800 CPU cycles (about 9.3 μs under 300 MHz and 4.2 μs under 667 MHz). This overhead is low and negligible relative to multimedia execution, since the decode time of one MPEG frame is about 5ms to 40ms and the decode time of one H.26x packet is about 1ms to 7ms under 667 MHz. It means that these prediction methods only incur about 0.0105% to 0.42% overhead.

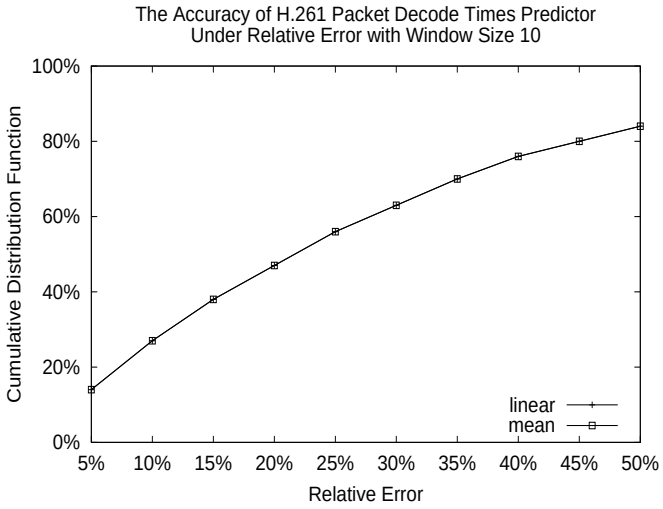
MPEG Frame Decode Time Predictor	2738
Video Conferencing Frame Interval Predictor	2560
Video Conferencing The Number of Packets in a Frame	68
Video Conferencing Packet Decode Time Predictor	2731

Table 10: Cost of Prediction Methods (in CPU cycles).

Finally, we measure the cost of voltage and frequency scaling. To do this, we adjusted the processor from one frequency to another frequency, and measure the number of cycles for each change. The results in Table 11 show that the CPU can change speed within 1125 cycles (about 3.75 μs under 300 MHz and 1.69 μs under 667 MHz). It means that the voltage and frequency



(a) CDF of Absolute Error



(b) CDF of Relative Error

Figure 16: The Accuracy of H.261 Packet Decode Times Predictor under Resolution 176x144

		to frequency (MHz)				
		300	400	533	600	667
from frequency (MHz)	300		1101	1099	1086	1066
	400	1125		1095	1086	1066
	533	1117	1104		1073	1066
	600	1125	1101	1092		1066
	667	1117	1101	1088	1077	

Table 11: Cost of Voltage and Frequency Scaling (in CPU cycles).

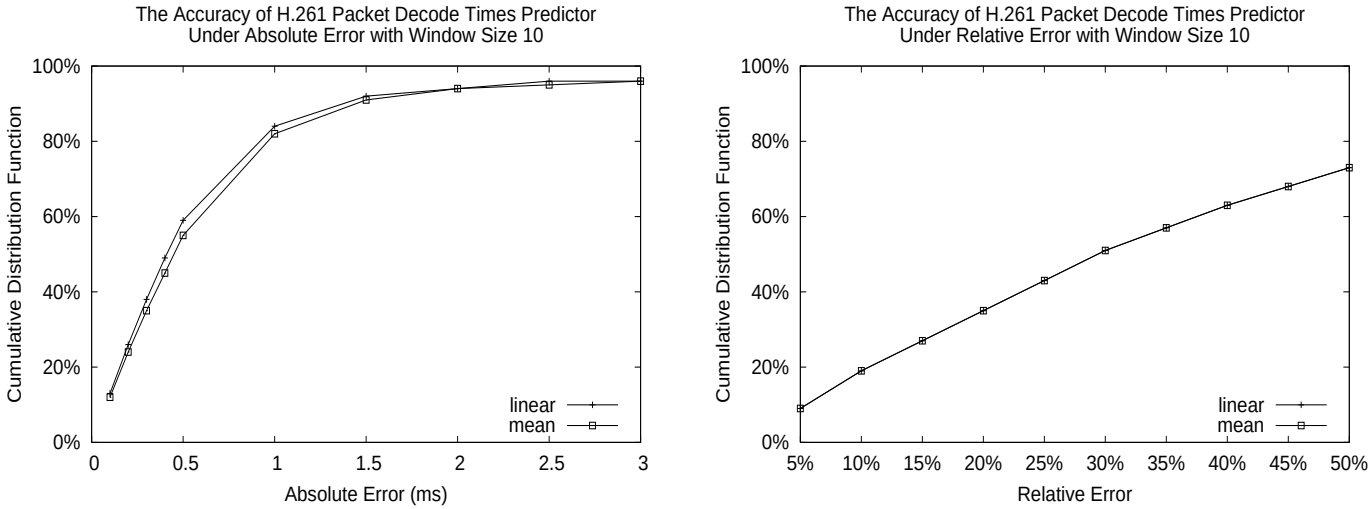
scaling only incurs tiny overhead.

6 Related Work

Recently, application aware/directed/controlled power management for processors has received increasing research attention and a variety of techniques have been proposed. Most of these techniques [5, 16, 22, 19, 17, 29] utilize the dynamic voltage and frequency scaling technique (DVFS) of processors for energy savings, while several other techniques [26, 25, 28] utilize the application/middleware based adaptation for energy savings of processors.

For instance, at the application/middleware based adaptation, Shenoy [28] and Tamai [25] both suggest performing power friendly proxy based video transformations to reduce video quality (i.e. bit-rate, resolution, fps) for energy savings; Flinn [26] utilizes Puppeteer [27], a component-based adaption system, to reduce document quality (i.e. picture resolution, color depth, animation) for energy savings of office applications. The common features of these application/middleware based adaptation techniques are: (i) the amount of to-do work of applications is reduced, for example, the video quality in [28, 25] and the document quality in [26] are reduced; (ii) all of them utilize a remote proxy to reduce the amount of to-do work of applications; (iii) all of them are designed to handle applications which request their data (i.e. video stream, audio stream) from remote servers via network, therefore they are not suitable for applications which only use local data.

In contrast, the application aware/directed/controlled power management techniques [5, 16, 22, 19, 17, 29] which utilize DVFS technique: (i) do not reduce the amount of to-do work of applications; (ii) do not require a remote proxy to transcode data; (iii) not only handle applications which use remote data but also handle applications which only use local data. In [5, 16, 19, 17, 29], researchers proposed several different application-controlled DVFS techniques for video decoder. For instance, Mesarina [17] proposed a offline algorithm to compute the order and voltage settings at which the appliance's CPU decodes the frames, reducing energy consumption without violating timing for buffering constraints. This technique requires each video being preprocessed offline by some server ahead of its energy efficient playback. While in [5, 16, 19, 29], several techniques were



(a) CDF of Absolute Error (b) CDF of Relative Error

Figure 17: The Accuracy of H.261 Packet Decode Times Predictor under Resolution 352x288

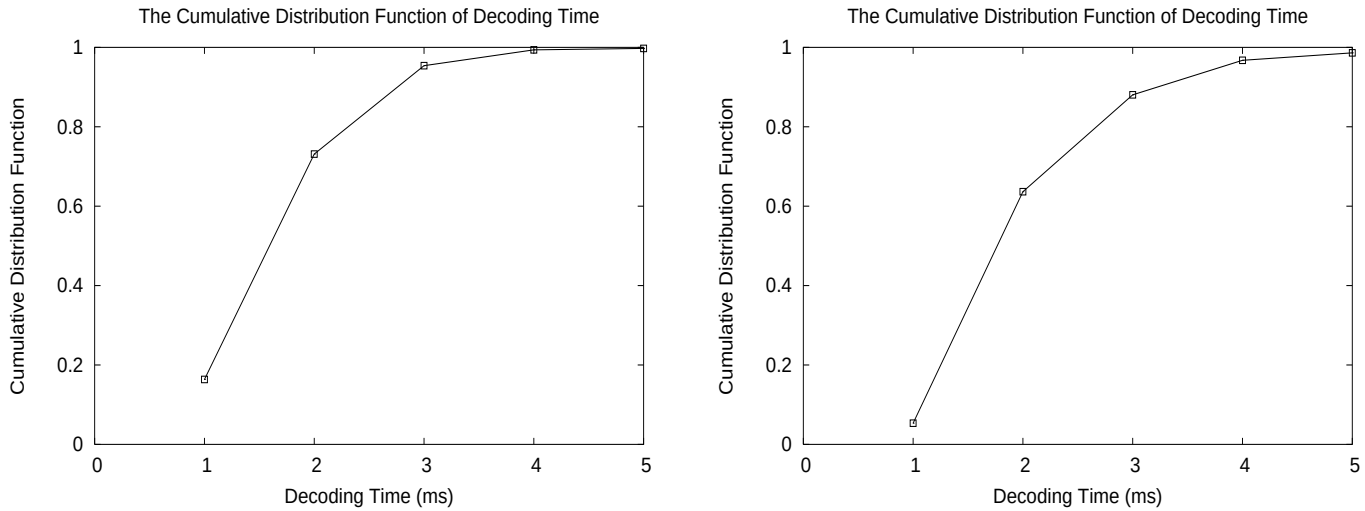
proposed to online estimate the CPU demand of video decoding, thus eliminated the need of preprocessing videos offline. All these techniques [5, 16, 19, 17, 29] only consider a single application, video decoder, and grant complete control of the processor frequency and voltage settings to the video decoder. The power-aware video decoder can choose a system-wide voltage and frequency setting based on its needs and typically ignores other applications in the system. As a result, the performance of other applications can be significantly impacted when settings chosen by the power-aware video decoder do not satisfy their CPU needs. The GRACE-OS project [22] proposed an application/OS cooperative approaches to achieve energy savings for periodic multimedia applications via DVFS technique, and they provided performance isolation among applications. GRACE-OS is not a general purpose approach, they can only handle periodic multimedia applications but not aperiodic applications such as web browser.

An integrated power management approach was proposed in [23] to unify low level architectural optimizations (CPU, memory, register), OS power-saving mechanisms (Dynamic Voltage and Frequency Scaling) and adaptive middle techniques (admission control, optimal transcoding, network traffic regulation). In this technique, interaction parameters between the different levels are identified and optimized to significantly reduce power consumption.

7 Summary and Conclusions

This paper proposed Chameleon, a new approach for power management in mobile processors. We argued that application know best what their energy needs are and propose an approach that puts the entire burden of power management on individual applications. The operating system only enforces protection and isolates applications from the power settings of other applications. Chameleon consists of three components: (i) a *common interface* that power-aware applications can use to measure their CPU demands and adjust their CPU speed settings correspondingly, (ii) a modified kernel CPU scheduler that supports per-process CPU speed setting to ensure performance isolation among applications, and (iii) a *speed adapter* that maps these CPU speed settings to the nearest speed actually supported by the hardware.

We implemented Chameleon in the Linux kernel and evaluated its energy efficiency for three time-sensitive applications, namely a video decoder, a video conferencing tool and a web browser. Our results show that Chameleon can extract up to 34% energy savings when compared to LongRun and up to 50% savings when compared to recently proposed OS-based DVFS techniques, while delivering comparable or better performance to time-sensitive applications. Chameleon is also more effective at scheduling a mix of concurrent application with diverse energy needs. As part of future work, we are developing a power-aware version of an Office application suite for Chameleon.



(a) 176x144 (b) 352x288

Figure 18: The CDF of Decoding Time for H.261 Standard

References

[1] A. Bavier, A. Montz, and L. Peterson. Predicting MPEG Execution Times. In *Proceedings of ACM Sigmetrics'98, Madison, WI*, pages 131–140, June 1998.

[2] G. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis Forecasting and Control Third Edition*. Prentice Hall, 1994.

[3] J. Bendat and A. Piersol. *Random Data Analysis and Measurement Procedures Second Edition*. John Willey & Sons, 1985.

[4] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[5] K. Choi, K. Dantu, W. Cheng, and M. Pedram. Frame-based Dynamic Voltage and Frequency Scaling for a MPEG Decoder. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design (CAD'02), San Jose, CA*, pages 732–737, November 2002.

[6] Dillo 0.7.3. Dillo Org., http://www.dillo.org.

[7] D. R. Engler, M. Kaashoek, and J. O. Jr. Exokernel: An Operating System Architecture for Application-level Resource management. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP95), Copper Mountain, CO*, pages 251–266, December 1995.

[8] K. Flautner, S. Reinhardt, and T. Mudge. Automatic Performance-setting for Dynamic Voltage Scaling. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom'01), Rome, Italy*, pages 260–271, July 2001.

[9] K. Flautner and T. Mudge. Vertigo: Automatic Performance-setting for Linux. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA*, pages 105–116, December 2002.

[10] M. Fleischmann. Longrun Power Management - Dynamic Power Management for Crusoe Processors. Technical report, Transmeta Corporation, 2001.

[11] Gnomemeeting 0.96.1. Gnomemeeeting Org., http://www.gnomemeeting.org.

[12] D. Grunwald, P. Levis, K. Farkas, C. M. III, and M. Neufald. Policies for Dynamic Clock Scheduling. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI'00), San Diego, CA*, pages 73–86, October 2000.

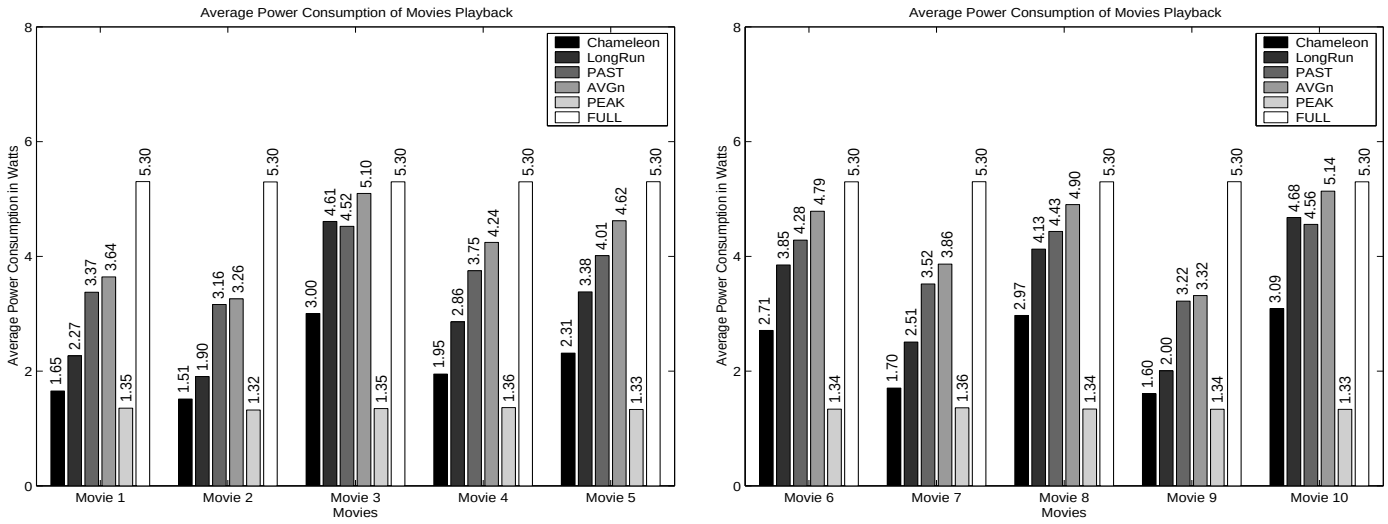


Figure 19: Average CPU Power Consumption during Video Playback

[13] K.Govil, E.Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-setting of a Low-power CPU. In *Proceedings of the 1st Mobile ACM/IEEE International Conference on Computing and Networking Conference (MobiCom'95), Berkeley, CA*, pages 13–25, November 1995.

[14] J. R. Lorch and A. J. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proceedings of the 2001 ACM SIGMETRICS Conference, Cambridge, MA*, pages 50–61, June 2001.

[15] J. R. Lorch and A. J. Smith. Operating System Modifications for Task-based Speed and Voltage scheduling. In *Proceedings of the 1st ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys'03), San Francisco, CA*, pages 215–229, May 2003.

[16] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads. In *Proceedings of the 3rd ACM/IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASE'01), Greenoble, France*, pages 156–163, October 2002.

[17] M. Mesarina and Y. Turner. Reduced Energy Decoding of MPEG Streams. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference (MMCN)*, pages 73–84, January 2002.

[18] Mplayer 0.90. http://www.mplayerhq.hu.

[19] J. Pouwelse, K. Langendoen, I. Lagendijk, and H. Sips. Power-aware Video Decoding. In *Proceedings of the 22nd Picture Coding Symposium (PCS'01), Seoul, Korea*, pages 303–306, April 2001.

[20] Crosoe TM5600 Processor Data Sheet. Transmeta Inc., http://www.transmeta.com.

[21] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI'94), Monterey, CA*, pages 13–23, November 1994.

[22] W. Yuan and K. Nahrstedt. Energy-efficient Soft Real-time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY*, pages 149–163, October 2003.

[23] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau and N. Venkatasubramanian. Integrated Power Management for Video Streaming for Mobile Handheld Devices. In *Proceedings of the 11th ACM Internatioanl Conference on Multimedia (MM'03), Berkeley, CA*, pages 582–591, November 2003.

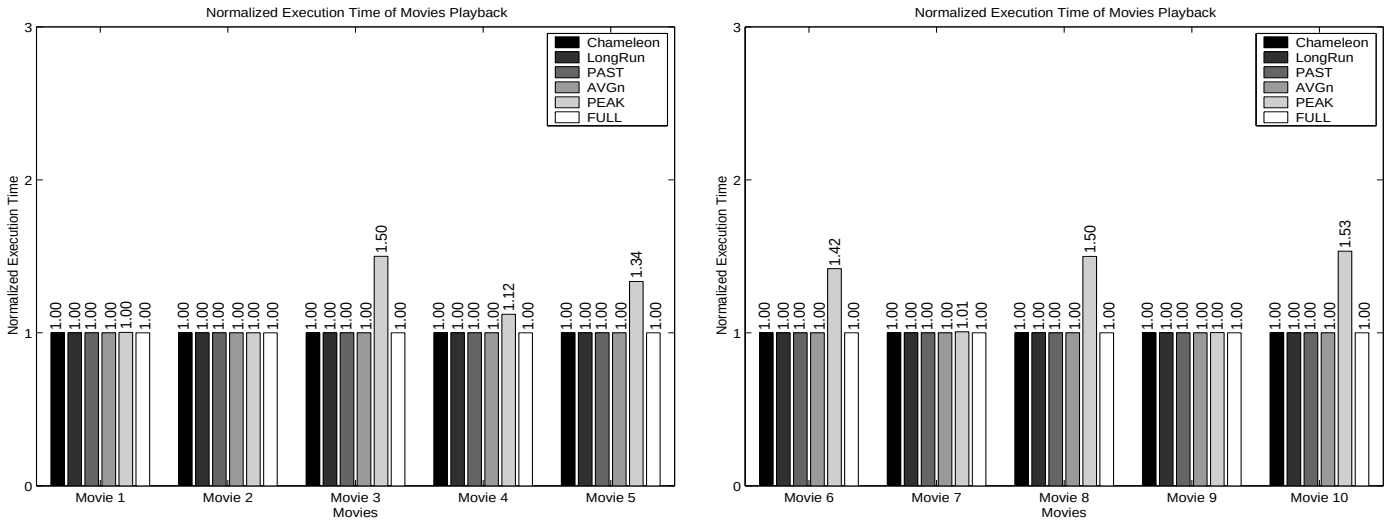


Figure 20: Normalized Execution Time during Video Playback

[24] X. Liu, P. Shenoy and W. Gong. A Time Series-based Approach for Power Management in Mobile Processors and Disks. In *Proceedings of the 14th ACM Internatioanl Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'04), Cork, Ireland*, pages 74–79, June 2004.

[25] M. Tamai, T. Sun, K. Yasumoto, N. Shibata and M. Ito. Energy-aware Video Streaming with QoS Control for Portable Computing Devices. In *Proceedings of the 14th ACM Internatioanl Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'04), Cork, Ireland*, pages 68–73, June 2004.

[26] J. Flinn, E. Lara, M. Satyanarayanan, D. Wallach and W. Zwaenepoel. Reducing the Energy Usage of Office Applications. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany*, November 2001.

[27] E. de Lara, D. Wallach and W. Zwaenepoel. Puppeteer: Component-based Adaptation for Mobile Computing. In *Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems (USITS 01), San Francisco, CA*, pages 159–170, March 2001.

[28] P. Shenoy and P. Radkov. Proxy-assisted Power-friendly Streaming to Mobile Devices. In *Proceedings of the 2003 Multimedia Computing and Networking Conference (MMCN'03), Santa Clara, CA*, pages 177-191, Janauary 2003.

[29] D. Son, C. Yu and H. Kim. Dynamic Voltage Scaling on MPEG Decoding. In *Proceedings of the 2001 International Conference of Parallel and Distributed System (ICPADS'01), KyongJu City, Korea*, pages 633–640, June 2001.

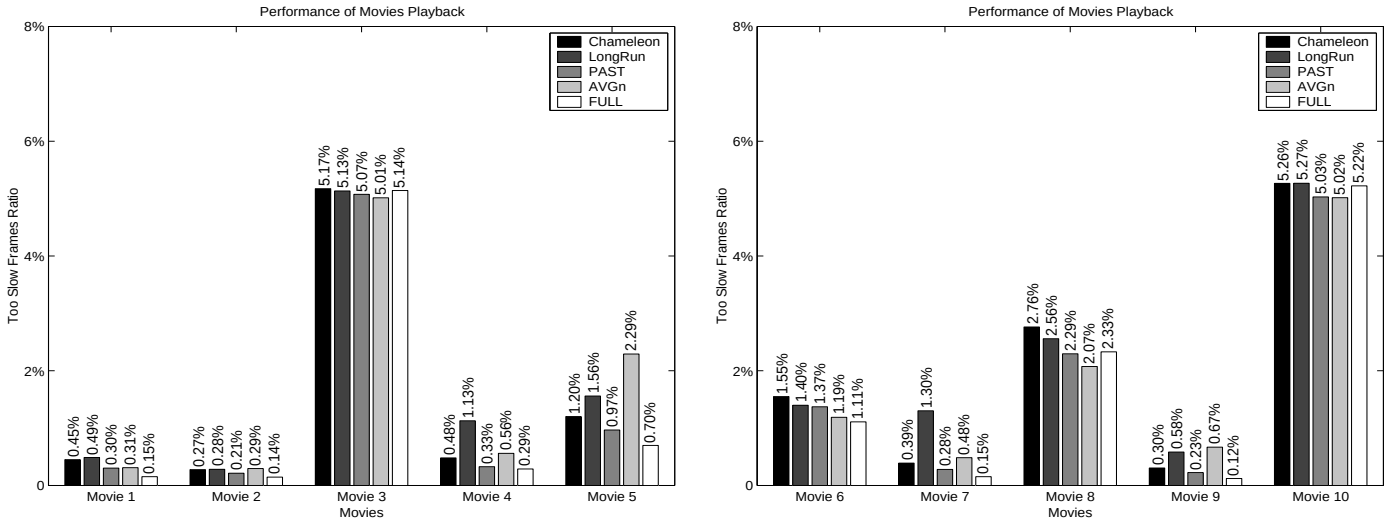


Figure 21: Late Frames Percentage during Movie Playback

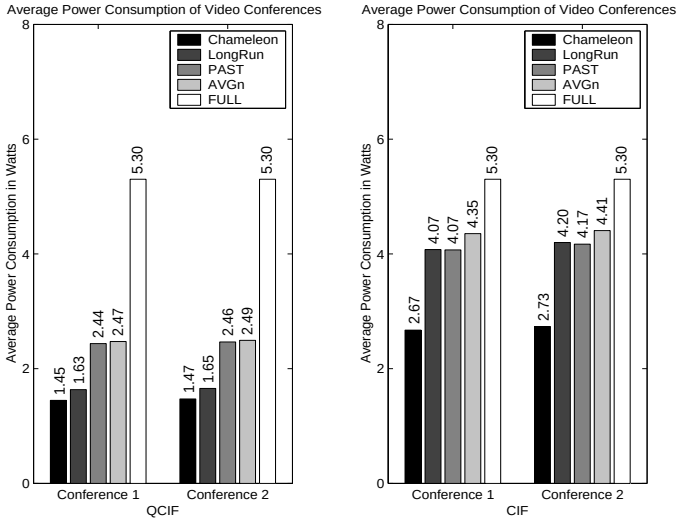


Figure 22: Average Power Consumption for Video Conferencing.