A Queuing Theoretic Approach to Processor Power Adaptation for Video Decoding Systems

Brian Foo and Mihaela van der Schaar, *Senior Member, IEEE*

***Abstract*—Video decoding applications must often cope with highly time-varying workload demands, while meeting stringent display deadlines. Voltage/frequency scalable processors are highly attractive for video decoding on resource-constrained systems, since significant energy savings can be achieved by dynamically adapting the processor speed based on the changing workload demand. Previous works on video-related voltage scaling algorithms are often limited by the lack of a good complexity model for video and often do not explicitly consider the video quality impact of various steps involved in the decoding process. Our contribution in this paper is threefold. First, we propose a novel complexity model through offline training that explicitly considers the video source characteristics, the encoding algorithm, and platform specifics to predict execution times. Second, based on the complexity model, we propose low complexity online voltage scaling algorithms to process decoding jobs such that they meet their display deadlines with high probability. We show that on average, our queuing-based voltage scaling algorithm provides approximately 10%–15% energy savings over existing voltage scaling algorithms. Finally, we propose a joint voltage scaling and quality-aware priority scheduling algorithm that decodes jobs in order of their distortion impact, such that by setting the processor to various power levels and decoding only the jobs that contribute most to the overall quality, efficient quality, and energy tradeoffs can be achieved. We demonstrate the scalability of our algorithm in various practical decoding scenarios, where reducing the power to 25% of the original power can lead to quality degradations of less than 1.0 dB PSNR.**

***Index Terms*—Complexity modeling, dynamic voltage scaling (DVS), priority scheduling, queuing theory, video decoding, waiting time analysis.**

I. Introduction

RECENTLY, many processors that support multiple operating frequencies have become commercially popular [1], [2]. Consequently, various dynamic voltage scaling (DVS) algorithms have been proposed for dynamically adjusting the operating frequency and voltage of a processor to utilize energy-delay tradeoffs for a task where jobs need to be completed by certain deadlines [3]. In CMOS circuits, power consumption is given by $Power \propto V^2 \cdot C_{\text{eff}\cdot\text{f}}$, where $V, C_{\text{eff},\text{f}}$ denote the voltage, effective capacitance and operating frequency, respectively. The energy spent on one task is proportional to the time spent for completing that task and time is inversely proportional to frequency. Hence, the energy is proportional to the square of the voltage, i.e., $Energy \propto V^2 C_{\text{eff}}$. The energy spent on one process can be reduced by decreasing the voltage, which will correspondingly increase the delay. Based on statistical estimates of the cycle requirement (i.e., complexity or execution time) for each job, a DVS algorithm assigns an operating level (i.e., power and frequency) for processing that job while meeting delay requirements for that job.

In the past few years, a wide variety of DVS algorithms have been proposed for delay-sensitive applications [4]–[11], [13], [14]. Some DVS algorithms perform optimization over only one or two tasks, such that the processor power level is determined on the fly to meet imminent (soft) deadlines while considering either the worst case execution time (WCET) [5], [6], or the average case execution time (ACET) [10]. While these approaches have very low computational complexity, the performances are limited in that future tasks with imminent deadlines may require extremely high processing power to finish in time after the completion of the current task. On the other hand, more robust DVS algorithms, such as the cycle-conserving and look-ahead earliest deadline first DVS [4], and Feedback Control-based DVS [7], schedule the power based on multiple future task deadlines. The complexity of such approaches can become huge for large job buffer sizes, since many job deadlines must be jointly considered in such scheduling schemes. This may often be the case for multimedia where video packet arrivals over a network are nondeterministic, and many packets are required to decode each video frame. Consequently, various lower-complexity DVS approaches were proposed, where the number of tasks released for execution (and hence, the number of deadlines to consider in the DVS algorithm) could be controlled by adjusting various parameters, such as the "aggressiveness" factor in [11].

In spite of the wide variety of algorithms proposed, current DVS approaches are limited in several ways.

- Current DVS algorithms lack simple yet accurate complexity models for multimedia tasks. Many DVS algorithms are often optimized in an application-agnostic or *ad hoc* manner, or otherwise they add significant overhead to online complexity adaptation [7], [11], [29], [31]. Other more formal, queuing-theoretic DVS approaches [15],

Manuscript received May 18, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Markus Pueschel. This work was supported by the grants NSF CCF 0541867 (Career Award), NSF CCF 0541453 and NSF 0509522 from the National Science Foundation. The material in this paper was presented in part at the IEEE International Conference on Image Processing (ICIP), San Antonio, TX, September 2007.

The authors are with the Department of Electrical Engineering, University of California Los Angeles (UCLA), Los Angeles, CA 90095 USA (e-mail: bkungfoo@ee.ucla.edu; mihaela@ee.ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2007.906737

[16], [43] for generic applications use models that are not well suited toward the highly time-varying video decoding complexity.
- Current DVS algorithms often use worst-case or average case complexity measurements (e.g., [5], [6], [10]), which neglect the fact that multimedia compression algorithms require time-varying resources that differ significantly between jobs. Moreover, "worst-case" and "average-case" metrics do not exploit the information stored by the second moment of job execution times, or by the execution time distributions themselves. As we will show in this paper, such information can often be used to perform smoother voltage scaling (i.e., fewer voltage switches) and improve performance.
- Current DVS algorithms do not cooperate with multimedia applications to obtain complexity statistics, which may vary across different coders, different sequences, and different bit rates.
- While the generic framework of imprecise computation has been considered as an approach for loss-tolerant applications such as multimedia [17], [18], these algorithms do not take full advantage of the properties of the multimedia algorithm to optimize the quality or energy savings by jointly adapt the power level and the workload. Moreover, there is either no explicit consideration of the distortion impact in loss tolerant multimedia processing, or else an unrealistic model is used for distortion.

The address the limitations above, in this paper, we make the following contributions.
- We propose a complexity model that not only explicitly considers coder operations and frame dependencies (i.e., task deadlines), but can also be characterized by only a few parameters. Importantly, we show that complexity statistics can be decomposed into the sum of complexity metrics that follow simple, well-known distributions. By using offline training sequences, we derive complexity distributions for different types of jobs (such as decoding different video frame types) for different types of sequences/scenes. The encoder/server can then adapt the decoder's complexity model online with very low transmission overhead whenever the sequence characteristics or coder parameters change [29]. Hence, unlike prior DVS work, where this information is known a posteriori and adapting the complexity statistics may take up to several seconds or minutes (depending on the encoding structure), the proposed decoding system can optimally plan its use of resources based on *a priori* transmitted complexity traffic characteristics.
- Based on the online complexity distribution adaptation scheme, we propose a new, queuing theoretic model driven DVS paradigm, where by using the complexity model, the processor can efficiently select from various job processing disciplines (e.g., earliest deadline first, quality-aware priority scheduling) and adapt the processor power accordingly. We show the advantages of this approach compared to previous works where execution times are guided by simple metrics such as worst case execution time.
- We propose a quality-aware DVS algorithm based on priority scheduling, where jobs are decomposed based on their dependencies and contributions to overall video quality, such that more important jobs are processed first. In this way, the video stream can be decoded at various quality levels given different power levels, even if the average power is insufficient for decoding all jobs before their deadlines. Hence, the quality-aware DVS algorithm can retain high quality even if the available energy is reduced significantly.

The paper is organized as follows. Section II reviews a look-ahead DVS approach introduced in [4], which will be compared against the queuing-model paradigm for DVS. Section III introduces a queuing model approach for deadline-driven DVS algorithms. Section IV introduces a quality-adaptive DVS via a priority scheduling approach, where more important jobs are processed first. Section V provides performance comparisons between the look-ahead DVS and our deadline-driven queuing-based DVS algorithms, and shows different average power and quality tradeoffs achieved by priority-based DVS. Finally, Section VI concludes our work.

II. Real-Time Look-Ahead DVS and Motivation for Queuing-Based DVS

A. Look-Ahead EDF DVS Algorithm

We begin by introducing a well-known real-time DVS algorithm called look-ahead earliest deadline first (laEDF), which has been shown to provide 20%–40% energy savings [4]. The laEDF DVS algorithm attempts to process tasks at the lowest frequencies possible based on a look-ahead technique that considers future computational requirements. Rather than using a high operating frequency to satisfy the WCET for all tasks released for execution, as in the cycle-conserving EDF DVS [4], the laEDF tries to defer jobs such that the minimum amount of work is done while ensuring that all future deadlines will still be met. Of course, this means that the processor may be forced to speed up when worst-case situations occur.

The laEDF algorithm is shown below. Given jobs j, $j = 1, \ldots, J$, and operating frequencies f_k, $k = 1, \ldots, K$, we denote C_j to be the estimated WCET of job j, τ_j the period of arrivals of the jobs of the same type as j, T_j the decoding deadline for job j, and $c_{left,j}$ the worst case remaining computation for job j. Note that in the laEDF algorithm, we have assumed that, due to the regular frame structure of encoded videos, decoding jobs arrive periodically with τ_j. However, depending on the decoding buffer size, the application delay requirements, and the structure of the encoded frames, the decoding deadlines for each job, T_j, may vary for different jobs.

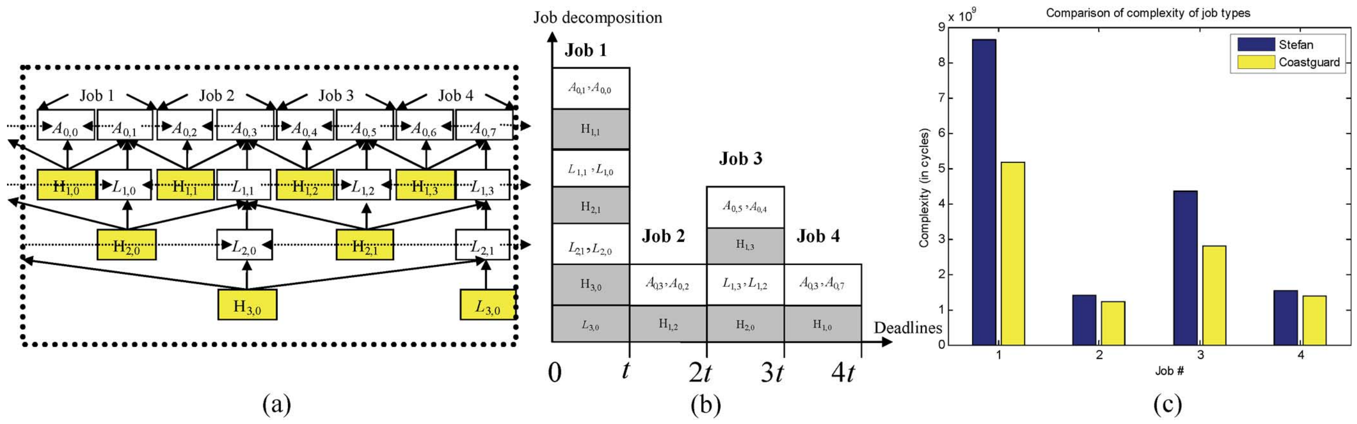


Fig. 1. (a) Periodic 3 temporal level MCTF structure (with dotted lines indicating motion compensation operations). (b) Job decomposition for each soft decoding deadline. (c) Corresponding workloads for the jobs in sequences *Stefan* and *Coastguard*.

laEDF DVS Algorithm:

```
select_frequency(x):
    use lowest freq. $f_i \in \{f_1, \dots f_K | f_1 < f_2 < \dots < f_K\}$
    such that $x < f_1/f_K$.
upon job_release($j$):
    set $c_{\text{left,j}} = C_j$;
    defer();
upon job_completion($j$)
    set $c_{\text{left,j}} = 0$
    defer();
during job_execution($j$):
    decrement $c_{\text{left,j}}$;
defer():
    set $U = C_1/\tau_1 + C_2/\tau_2 + \dots + C_J/\tau_J$;
    set $s = 0$;
    for $j = J$ to 1, step $-1$ (where $T_1 < T_2 < \dots < T_J$)
    /$*$ Note: reverse EDF order of tasks $*$/
    set $U = U - C_j/\tau_j$;
    set $x = \max(0, c_{left,j} - (1 - U)(T_j - T_1))$;
    set $U := U + (c_{left,j} - x)/(T_j - T_1)$;
    set $s := s + x$;
select_frequency $(s/(T_1 - \text{current\_time}))$;
```

B. Limitations of laEDF DVS and Motivation for Queuing-Based DVS

In this section, we describe in detail and give examples of some limitations of laEDF DVS. It was shown in [28] that power is a convex function of operating frequency, and therefore the total energy is minimized when the processor runs at a near constant power due to Jensen's inequality. However, one limitation of laEDF DVS is that it does not perform well when there are significant variations in complexity between jobs [4], since the power must be adjusted aggressively to accommodate worst-case scenarios. Unfortunately, sophisticated video coders have complex encoding structures and sequence characteristics, which lead to highly varying complexities between different jobs. To illustrate this, we consider an example from a motion compensation temporal filtering (MCTF) based video coder [33], which adopts a complex temporal prediction structure as shown in Fig. 1(a). Such sophisticated structures are very common in most current state-of-the-art video coders, as they can effectively exploit existing correlations among frames. However, this leads to groups of frames that need to be jointly decoded at the same time, and thus a burstier complexity traffic characteristic [see Fig. 1(b)]. For example, reconstructing the frame pair $\{A_{0,0}, A_{0,1}\}$ (i.e., job 1) in a 3–temporal level MCTF structure involves decoding or reconstructing the set of frames $\{L_{3,0}, \mathrm{H}_{3,0}, \mathrm{H}_{2,1}, \mathrm{H}_{1,1}, L_{2,1}, L_{2,0}, L_{1,1}, L_{1,0}, A_{0,1}, A_{0,0}\}$ (other required frames are already decoded for jobs in the previous GOP). On the other hand, reconstructing the subsequent frame pair $\{A_{0,2}, A_{0,3}\}$ (i.e., job 2) requires only decoding or reconstructing the set $\{H_{1,2}, A_{0,3}, A_{0,2}\}$, since other required frames were already previously decoded. To provide better intuition, we also decoded several sequences and various bit rates on a Pentium 4 processor using Windows XP safe mode with only a command prompt to ensure that context switches and interrupting processes were kept at a minimum. In Fig. 1(c), we measured the actual decoding complexity (workload) for each type of job in a group of frames of the *Coastguard* and *Stefan* sequences at the same bit-rates. Note that even within the same job type for a particular coder, different sequences can lead to significantly different complexities, i.e., jobs that require reconstructing many frames in high motion sequences such as *Stefan* may have much higher complexity than the corresponding jobs for *Coastguard*. Hence, even if laEDF DVS employs different WCET estimates for different types of

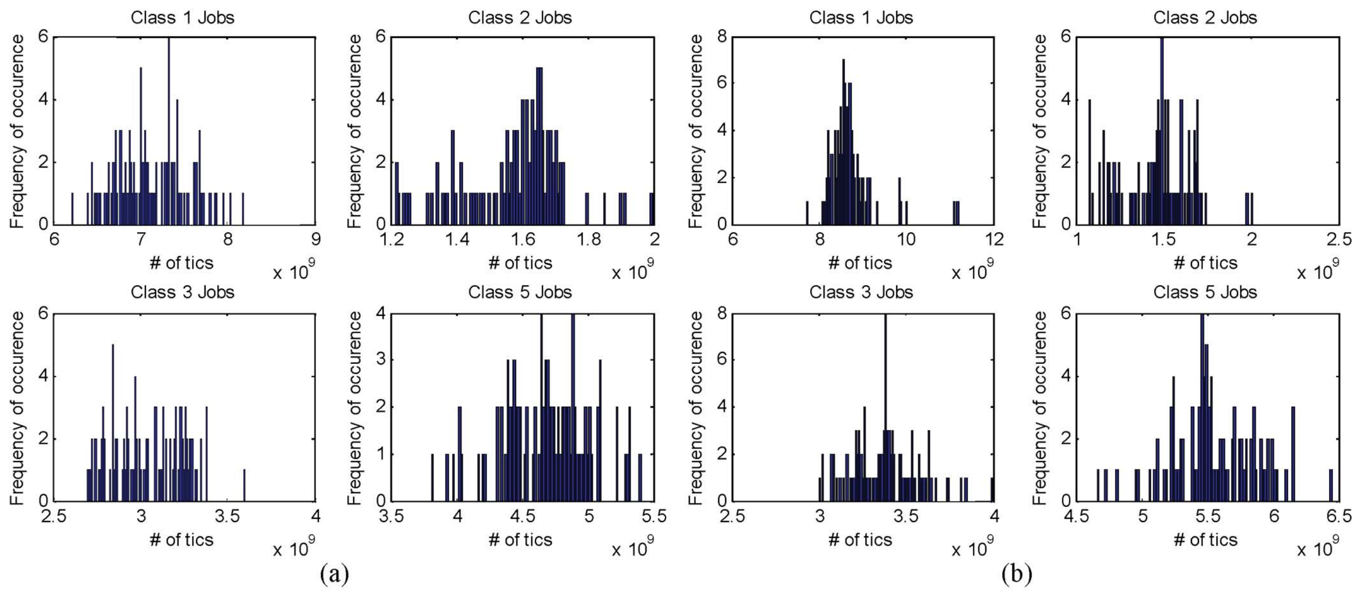


Fig. 2. The total service time distribution for various classes of jobs (out of a total of 8 classes) in 4 temporal level MCTF for (a) *Coastguard* and (b) *Stefan* sequences, decoded at 1152 kb/s.

jobs (e.g., the decoding of an I-frame, B-frame, or P-frame in MPEG), rapidly changing video content may force the laEDF to produce large processor power fluctuations during decoding, which is suboptimal in terms of energy savings.

We collected job execution times (offline) from a set of 11 training sequences with 16 GOPs each, decoded at seven different bit rates. As shown in Fig. 2(a) and (b), the complexity distributions for decoding different sequences shared similar features, such as the existence of peaks. (Here, tics correspond to the number of times an internal clock counter in the CPU is incremented, and can be converted to the number of CPU cycles through a constant scaling factor.) Nevertheless, the shapes vary greatly between different sequences, as shown in the figure for *Coastguard* and *Stefan*.

A final limitation is that laEDF DVS does not consider the loss-tolerant nature of multimedia, and therefore does not adapt gracefully to different levels of video quality and power whenever jobs are dropped. For example, in Fig. 1(a), the level 3 L-frame is required to decode all A-frames in the GOP, while lower level L-frames and H-frames are required to decode fewer frames. Hence, some of the lower level frames can be discarded with less impact on the quality of the video sequence compared to discarding the highest level L-frame. The order in which these frames are decoded forms a basis for a quality-adaptive DVS architecture, which will be discussed in Section IV.

In the next section, we present a complexity modeling approach that captures time-varying distributions at a fine granular level. In addition, a DVS approach based on delay deadline-driven queuing theory is introduced, where power is adapted based on the proposed complexity model to meet "hard" deadlines with high probability. In Section IV, we extend this queuing model to include priority-scheduling of jobs, such that less important jobs may be discarded without significant degradation to the video quality.

III. Deadline-Driven Queuing-Based DVS Complete to Overcomplete Representations

In this section, we introduce a queuing theoretic approach to determine the probability of missing job delay deadlines given a processor frequency level and show how this can be used to derive efficient DVS policies. In order to use queuing theory however, we must first derive a stochastic complexity model that is practically accurate.

A. *Challenges and Previous Works for Complexity Modeling*

Modeling the complexity of state-of-the-art video coders in a both accurate and elegant way is a challenging task due to the complex group-of-pictures (GOP) structures that exist, where many neighboring video frames are coded together. In addition, some advanced coders (e.g., MPEG4) allow the GOP structure to change over time to adapt to changing video source characteristics, in which case the complexity model must adapt by recapturing statistics whenever the GOP structure changes. As a result of these complex and potentially changing encoding structures, research on complexity prediction and modeling have traditionally fallen into two categories. The first category involves methods that ignore coder-specific operations, such as coarse levels of empirical modeling for complexity [31], or the use of a statistical sliding window [5]. The second category involves modeling complexity at a fine granular level based on functions associated with the process of decoding (e.g., entropy decoding, inverse transform, etc.). However, these works do not provide theoretical models [33], or else they are based on platform-independent "virtual" complexities that can not be mapped into real complexity (time) in a straightforward manner [21], [32]. In the following section, we propose a mixed modeling technique, where decoding complexities are measured at a fine-granular level and then modeled by well-known distributions.

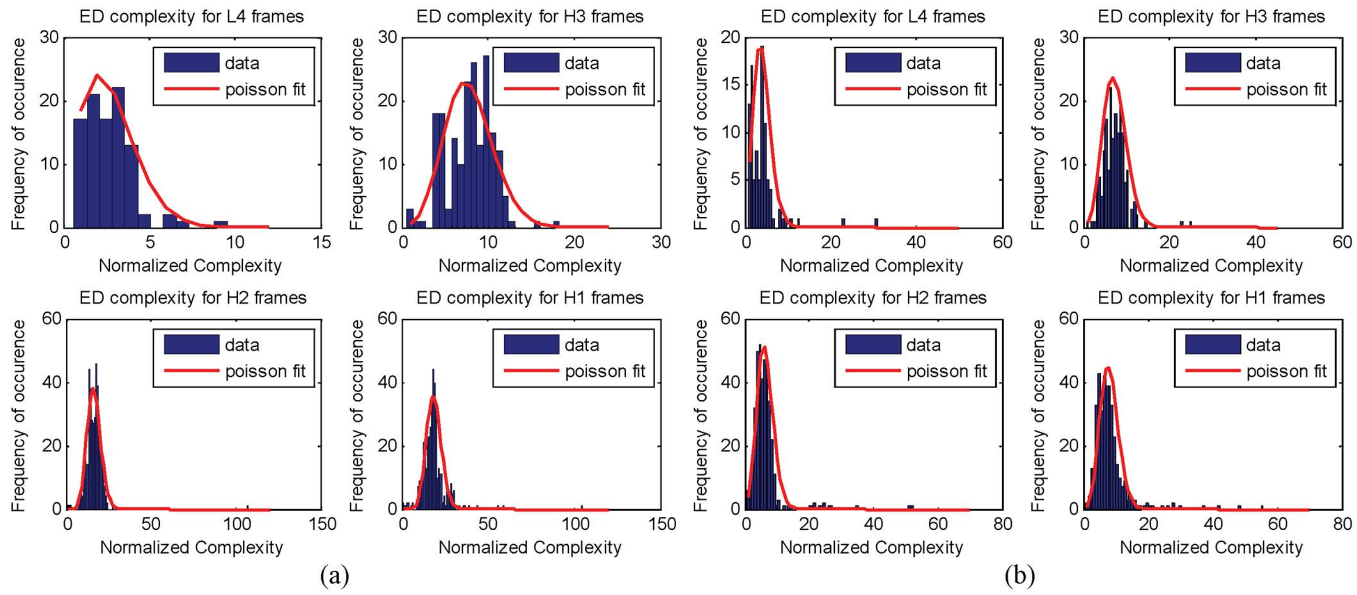


Fig. 3. Normalized entropy decoding complexity for various L- and H-frames in a 4 temporal level MCTF GOP for (a) *Coastguard* and (b) *Stefan* sequences, decoded at 1152 kb/s.

B. Deriving and Modeling the Service Distribution for Jobs Properties of the Derived Formulation

Based on our discussion earlier on mixing training data with analytical models, we show an example using an MCTF coder with four temporal levels. In order to differentiate between various jobs associated with each GOP, we define job classes $i = 1, \ldots, I$, where a job belongs to class i if it is the ith job to be decoded in its associated GOP. For example, in the MCTF structure shown in Fig. 1, there are a total of 4 classes of jobs which correspond to the decoding of the $\{A_{0,0}, A_{0,1}\}$, $\{A_{0,2}, A_{0,3}\}$, $\{A_{0,4}, A_{0,5}\}$, and $\{A_{0,6}, A_{0,7}\}$ frames in a GOP. For 4 temporal levels, we have eight job classes. It is important to classify these jobs in such a way because jobs in the same class are expected to have similar complexities, and similar waiting times before being processed.

We collected job execution times (offline) from a set of 11 training sequences with 16 GOPs each, decoded at seven different bit rates. In Fig. 2(a) and (b), we show the complexity distributions (in tics) for various job classes, averaged over all sequences, decoded at bit rates 1152 and 320 kb/s. Here, tics are the number of times an internal clock counter in the CPU is incremented, and can be converted to the number of CPU cycles through a constant scaling factor. We noticed that the complexity distributions shared similar features, such as the existence of peaks. We also explored the complexity distribution dependency on sequences by collecting data from different classes of jobs for particular sequences over seven different bit rates (from 200 kb/s to 1.5 mb/s) and normalizing the measurements by their scales in order to obtain an average distribution shape for each sequence. It was discovered that the shapes vary greatly between different sequences, as shown in the comparison of the sequences *Coastguard* and *Stefan* in Fig. 2(c) and (d).

In order to better model the complexity analytically, we investigated the complexities contributed by different steps of a decoding process. Decoding jobs often involves multiple different functions such as entropy decoding (ED), inverse transform (IT), motion compensation (MC), and fractional pixel interpolation (FI). Hence, the total complexity for class i for a sequence seq, C_i^{seq}, is the sum of complexities associated with each of the various decoding functions:

$$C_i^{\text{seq}} = C_{i,\text{ED}}^{\text{seq}} + C_{i,\text{IT}}^{\text{seq}} + C_{i,\text{MC}}^{\text{seq}} + C_{i,\text{FI}}^{\text{seq}} \tag{1}$$

where each $C_{i,\text{op}}^{\text{seq}}$ indicates the total complexity associated with one type of decoding step for a job of class i. Since a job of class i is composed of decoding and reconstructing various frames at various temporal levels, we can further decompose $C_{i,\text{op}}^{\text{seq}}$, $op \in \{\text{ED}, \text{IT}, \text{MC}, \text{FI}\}$, into a sum of decoding steps performed at each temporal decomposition level. For example, ED complexity of a class 1 job in 3-level MCTF, as shown in Fig. 1, consists of entropy decoding frames $\{L_{3,0}, \text{H}_{3,0}, \text{H}_{2,1}, \text{H}_{1,1}\}$. Likewise, for Ω temporal level MCTF, the ED complexity for a class 1 job can be expressed as the sum of complexities for all of its entropy decoding tasks:

$$C_{1,\text{ED}}^{\text{seq}} = C_{L(\Omega),\text{ED}}^{\text{seq}} + \sum_{\omega=1}^{\Omega} C_{\text{H}(\omega),\text{ED}}^{\text{seq}} \tag{2}$$

where $C_{fr(\omega),\text{ED}}^{\text{seq}}$ is the entropy decoding complexity of decoding a frame of type fr at temporal level ω. We can finally model $C_{fr(\omega),\text{op}}^{\text{seq}}$ with simple distributions that require only a few parameters, and sum up the distributions to form C_i^{seq}. A particular interesting example comes from entropy decoding, where the normalized complexity distribution $\hat{C}_{fr(\omega),\text{ED}}^{\text{seq}}$ is Poisson, i.e.,

$$p_{fr(\omega),\text{ED}}^{\text{seq}}(n) = \frac{\left(\nu_{fr(\omega)}^{\text{seq}}\right)^n e^{-\nu_{fr(\omega)}^{\text{seq}}}}{n!} \tag{3}$$

where n is a Poisson bin number, $p_{fr(\omega),\text{ED}}^{\text{seq}}(n)$ is the probability that the normalized complexity falls into bin n, and $\nu_{fr(\omega)}^{\text{seq}}$

TABLE I
AFFINE TRANSFORM COEFFICIENTS FOR THE ENTROPY DECODING COMPLEXITY IN 4-LEVEL MCTF

Frame	All Sequences (1152kbps)			Coastguard			Stefan		
	a_i^{seq}	b_i^{seq}	n_i^{seq}	a_i^{seq}	b_i^{seq}	n_i^{seq}	a_i^{seq}	b_i^{seq}	n_i^{seq}
L4	4.4e7	8.7e8	21	53e6	2.5e8	2.6	18e6	5.9e8	4.0
H3	1.1e7	0.9e8	9.4	16e6	1.5e8	7.8	7.0e6	2.7e8	7.4
H2	1.7e7	2.0e8	13	7.4e6	1.1e8	16	8.7e6	2.2e8	6.3
H1	3.1e7	4.8e8	8.7	5.0e6	0.9e8	19	7.6e6	1.5e8	8.1

is a shape parameter for the normalized complexity distribution. The real entropy decoding complexity distribution can be modeled by a shifted and scaled Poisson distribution, i.e.,

$$C_{fr(\omega),\mathrm{ED}}^{\mathrm{seq}} \cong a_{fr(\omega)}^{\mathrm{seq}} \hat{C}_{fr(\omega),\mathrm{ED}}^{\mathrm{seq}} + b_{fr(\omega)}^{\mathrm{seq}} \quad (4)$$

where $a_{fr(\omega)}^{\mathrm{seq}}$ and $b_{fr(\omega)}^{\mathrm{seq}}$ are sequence and frame dependent constants. Fig. 3 shows the normalized ED complexities $\hat{C}_{fr(\omega),\mathrm{ED}}^{\mathrm{seq}}$ for various L-frames and H-frames of the sequences *Coastguard* and *Stefan* decoded at 1152 kb/s. Notice that the distributions in Fig. 3 denote a subset of the complexities forming the distributions for different classes of jobs in Fig. 2 using (4), (2), and (1). Table I shows the coefficient values for the normalized entropy decoding complexity distribution averaged over all training sequences, along with individual sequences *Coastguard* and *Stefan*. Note that in all three cases, the coefficients a_i^{seq}, b_i^{seq}, and ν_i^{seq} vary significantly for different sequences. Hence, in practice, coefficients need to be estimated separately for different sequences. The same modeling technique may be applied to inverse transforms and motion compensation.

While modeling complexity in this fashion is highly source dependent, our novelty lies in the low complexity and high accuracy of updating the complexity model whenever changes occur in the video source statistics. During long video sequences, advanced coders may change GOP sizes, coding rates, and frame sizes many times due to time-varying source statistics. Without a good complexity model, only loose bounds for complexity can be derived based on coarse parameters, such as the mean and variance of frame sizes and coding bit rates across entire sequences [34]. However, using our complexity model, the encoder can update the decoder's information of the video source by sending only a few distribution parameters, and the decoder can use these parameters to form accurate complexity distributions. (The reader is referred to [29] for more details on possible implementations of how these parameters can be efficiently transmitted to the decoder.)

Before moving into the queuing theoretic DVS, we make an important remark about the complexity model. The complexity distributions, which are measured and fitted to well-known distributions, are also based on the algorithmic decoder operations. As shown in the above example, entropy decoding complexity follows a simple shifted and scaled Poisson distribution, which is the limiting distribution when there are only a few high complexity tasks and many low complexity tasks. Indeed, algorithmically, entropy decoding for video frames follows such a distribution, since only a few decoded bits are used to reconstruct complex coder structures such as zero-trees, while most decoded bits are used to decode and refine significant coefficients [19], [20]. On the other hand, inverse transform followed a nearly constant complexity due to the particular implementation of the coder. Bidirectional motion compensation complexity led to the existence of two peaks, which occurred due to various macroblocks that required either a single prediction, or two predictions. Due to the space constraint in our manuscript, we omit a thorough analysis of each of these distributions. However, it should be noted that there is good reason to analytically model complexity based on these specific distributions, since they capture the underlying decoder implementation.

C. Delay Analysis

In this section, we assume that a buffer is available for receiving traffic from the network. Based on the job deadlines, the buffer periodically provides the decoder with jobs in the order of their display deadlines. We deploy a $D/G/1$ cyclic multiclass queuing system with single-service discipline for modeling a deadline-driven DVS system as shown in Fig. 4. Cyclic scheduling with single service discipline is also known as round-robin scheduling, where a job in class $i+1$ is always serviced directly after a job in class i, and a job of class 1 (in the next GOP) is always serviced directly after a job in class I (of the current GOP). The service policy for each class in this system can also be viewed as a single service discipline with a vacation period [24]. In particular, whenever a job of class i finishes service, the processor "goes on vacation" by servicing one job in each of the other classes, and "returns to service" to class i only after it has completed processing the jobs in other classes. Based on this service discipline and class-dependent service time distributions obtained through offline training or from an analytical model (Section III-C), we can determine the delay distribution for each class of jobs (Fig. 4). The delays can then be used to drive power scheduling to ensure that jobs are decoded before their deadlines.

In order to analyze the queuing system, we divide the interarrival time τ between jobs into N equally spaced time intervals, thereby obtaining discrete units of time τ/N s apart. Hence, each interarrival period contains time indices $n \in \{0, 1, \ldots, N-1\}$. Let $S_{i,k}(m)$ be a random variable representing the discrete service time of the ith job of the mth GOP at processor speed f_k, and $V_{i,k}(m)$ the following vacation period. Let $W_{i,k}(m)$ be the waiting time for the ith job of the mth GOP. The service, vacation, and waiting times are shown for a two jobs GOP structure in Fig. 5.

We suggest two methods for computing the delay distribution of jobs. The first method is based on the vacation time distribution. Suppose the processor runs at a constant speed f_k until steady state is reached. We denote the steady state (or time average) random variables as $S_{i,k}$, $V_{i,k}$, and $W_{i,k}$ with distributions $s_{i,k}(n)$, $v_{i,k}(n)$, $w_{i,k}(n)$. Waiting times approximations for a $D/G/1$ cyclic service queue and $D/G/1$ queues with vacations have been analyzed extensively [21]–[23]. For example,

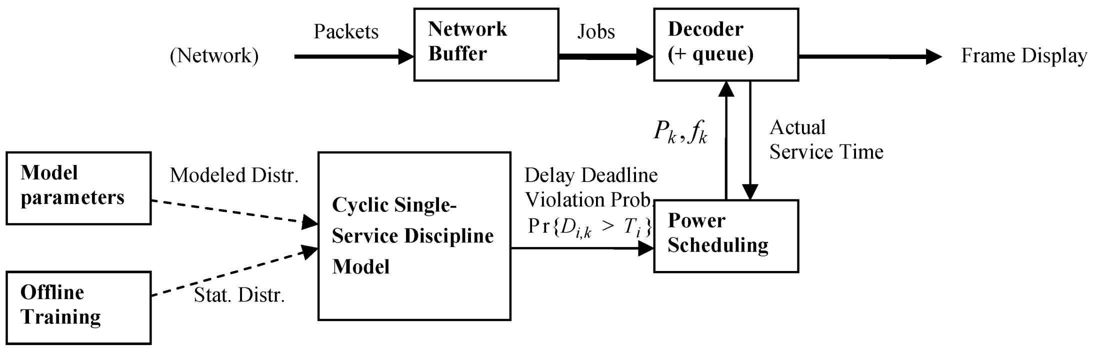


Fig. 4. Queuing model for deadline driven DVS.

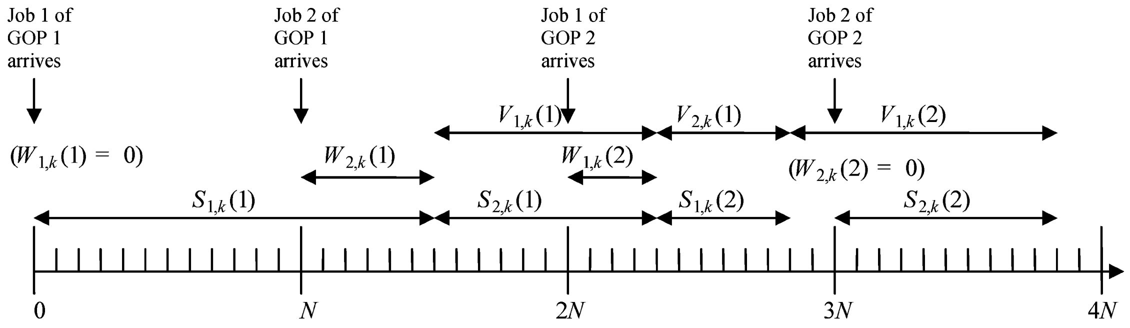


Fig. 5. An example of discrete service, vacation, and waiting times for cyclic service and 2 classes per GOP. Jobs of each class arrive at multiples of $2N$, with the first job of class 1 arriving at time 0.

given a discrete service time distribution $s_{i,k}(n)$ and vacation time distribution $v_{i,k}(n)$, the average waiting time is [23]

$$E[W_{i,k}] = \frac{H''_{i,k}(1)}{2H'_{i,k}(1)} + \sum_{r=1}^{N-1} \frac{1}{1-z_r} - \frac{N(N-1) - \left[G'_{i,k}(1) + 2G'_{i,k}(1)H'_{i,k}(1) + H''_{i,k}(1)\right]}{2\left[N - G'_{i,k}(1) - H'_{i,k}(1)\right]} \tag{5}$$

where $G_{i,k}(z)$ and $H_{i,k}(z)$ are the z-transforms of $s_{i,k}(n)$ and $v_{i,k}(n)$, respectively, and z_r are the unique roots of the polynomial $z^N - G_{i,k}(z)H_{i,k}(z)$ that are on or inside the unit circle, except $z = 1$.

The second method, which is shown below, is an analytical queuing approach based on service times directly. Without loss of generality, assume that we have the waiting time distribution for class 1 jobs, $w_{1,k}(n)$. The total delay of class 1 jobs, $D_{1,k} = S_{1,k} + W_{1,k}$, follows the distribution:

$$d_{1,k}(n) = w_{1,k}(n) * s_{1,k}(n) \tag{6}$$

where $*$ indicates convolution. Now, define $\tilde{D}_{1,k}$ for class 1 to have the following distribution:

$$\tilde{d}_{1,k}(n) = \begin{cases} d_{1,k}(n), & n > N \\ 1/(1-\alpha_1), & n = N \\ 0, & n < N \end{cases} \tag{7}$$

where

$$\alpha_1 = \Pr\{D_{1,k} > N\}. \tag{8}$$

Intuitively, $\tilde{D}_{1,k}$ is the delay of a class 1 job, conditioned on the total time being greater than N, which is when job 2 arrives. Hence, this conditional delay also defines the waiting time for the following class 2 job, i.e.,

$$w_{2,k}(n) = \tilde{d}_{1,k}(n-N). \tag{9}$$

For all other classes $i = 2, \ldots, I$, define the following distributions in the same way:

$$\tilde{d}_{i,k}(n) = \begin{cases} d_{i,k}(n), & n > iN \\ 1/(1-\alpha_i), & n = iN \\ 0, & n < iN \end{cases} \tag{10}$$

where

$$\alpha_i = \Pr\{D_{i,k} > N\} \tag{11}$$

and

$$w_{i,k}(n) = \tilde{d}_{i-1,k}(n-N). \tag{12}$$

The final result $w_{1,k}(n) = \tilde{d}_{I,k}(n-N)$ defining a recursive relationship for $w_{1,k}(n)$. In fact, the vacation time $v_{i,k}(n)$ can also be expressed by its transform

$$H_{i,k}(z) = \frac{\Lambda_{i-1,k}(z)}{z^{(I-1)N}\Psi_{i,k}(z)G_{i,k}(z)} \tag{13}$$

where $\Lambda_{i-1,k}(z)$ and $\Psi_{i,k}(z)$ are the z-transforms of $d_{i-1,k}(n)$ and $w_{i,k}(n)$. Because we are mainly interested in the probability that the waiting time exceeds some time t, we refer to [35] and [36] for the waiting time tail approximation given

$$\Pr\{W_{i,k} > t\} = \rho_k \exp\left(-\frac{\rho_k t}{\mathrm{E}[W_{i,k}]}\right) \tag{14}$$

TABLE II
OBTAINING THE WAITING TIME DISTRIBUTION AND PROBABILITY OF VIOLATING A DELAY DEADLINE

```
1.  Initialize $W_{1,k}^{(0)} = 0$, $iter = 0$.
2.  Do {
3.    Calculate $\tilde{s}_{2,k}^{(iter)}(n)$, $\tilde{s}_{2,k}^{(iter)}(n)$, ..., $\tilde{s}_{2,k}^{(iter)}(n)$ from (7) and (10), $n$ £ $N_{\max}$.
4.    Set $w_{i,k}^{(iter)}(n) = \tilde{s}_{i-1,k}^{(iter)}(n)$ for $i = 2,\ldots,I$.
5.  } while ($\left|E[W_{i,k}^{(iter)}] - E[W_{i,k}^{(iter-1)}]\right| > d$);
6.  Apply (16) to $E[W_{i,k}^{(iter)}]$ to determine tolerable delay under error prob. $e_i$.
```

where

$$\rho_k = \sum_{i=1}^{I} \frac{E[S_{i,k}]}{N} \tag{15}$$

is the average load on the system.

The waiting time tail approximation can be derived based on the approach shown in Table II, which iterates through (7) and (10) until the expected waiting time converges. The distributions are truncated at a sample size $N_{\max}$ under which the expected waiting time will be accurate. Because the waiting time tail distribution is truncated by $N_{\max}$, we use the approximation in (14) to estimate the waiting time tail distribution. The complexity of the waiting time estimation is proportional to $I \cdot N_{\max} \cdot N \cdot iter$, where I is again the total number of cyclic classes, and $iter$ is the number of iterations through the algorithm in Table II.

The last step of the algorithm in Table II is used to estimate waiting time tail distributions. However, the tail distribution of $D_{i,k}$ is of greater importance to our system, since the delay determines the probability that a hard deadline at time $t + T_i$ is missed for a job arriving at time t. However, since the tail distribution for $W_{i+1,k}$ has the same shape as the tail distribution of $D_{i,k}$ shifted by $-N$ in time [as can be shown from the relationship in (12)], we can apply the waiting time tail approximation in (14) to estimate the probability of violating delay deadline T_i

$$\begin{aligned}\Pr\{D_{i,k} > T_i\} &= \Pr\{W_{i+1,k} > T_i - N\} \\ &\approx \rho_k \exp\left(-\frac{\rho_k(T_i - N)}{\mathrm{E}[W_{i+1,k}]}\right).\end{aligned} \tag{16}$$

D. Queuing-Based DVS Scheduling Algorithms

In this section, we introduce a cross-layer optimization approach between the system and application based on queuing models. The processor captures its service time and waiting time distributions for different classes of jobs at various power levels P_k according to the algorithm described in Section III-C. The decoding system then produces a lookup table containing the net load ρ_k and expected delay $E[D_{i,k}]$ for each class i. Using (16), the processor can quickly estimate the probability of violating delay deadlines T_i. Finally, the video application sets upper bounds ε_i for the probability of dropping class i jobs, and determines a power schedule such that $\Pr\{D_{i,k} > T_i\} < \varepsilon_i$. Note that ε_i can be set to vary depending on the distortion impact of the respective job class. For example, job 1 in Fig. 1 may be considered more important than job 2, since many frames decoded in job 1 are also required to process job 2. We explore two statistical DVS optimization problems based on power optimization via queuing delay analysis, and provide two variations of an adaptive DVS algorithm based on these problems.

Optimization Problem 1: Minimizes the Average Power given Bounds on the Fraction of Dropped Jobs.

$$\begin{aligned}&\min_{k_i, i=1,\ldots,I} \sum_{i=1}^{I} E\left[S_{i,k_i}\right] P_{k_i} \\ &s.t.\ \Pr\left\{D_{i,k_i} > T_i\right\} < \varepsilon_i \\ &i \in \{1,2,\ldots,I\} \\ &k_i \in \{1,2,\ldots,K\}.\end{aligned} \tag{17}$$

Note that this problem implicitly considers sleep modes, where static power may be turned off during idle periods. Due to the delay constraints and static power, the problem is not convex, thus the solution has complexity $O(K^I)$. However, since video decoding is computationally very complex [29], [30], we assume that the static power consumption can be neglected due to very high active power consumption. When delay bounds are loose (e.g., T_i is large), we expect the queue to be almost never empty, and therefore the average processing time per job is close to the constant arrival rate τ. As mentioned in [28], for a fixed time τ to complete c cycles, the optimal energy saving schedule is to run the processor at a minimal constant speed, which is $f^* = c/\tau$. For discrete frequency levels, f^* can be achieved through timesharing between two adjacent frequency levels. While the delay constraints do not allow the processor to always run at the optimal average power level, the processor may usually run at a constant power level, and occasionally increase to higher power levels when the queue size (and aggregate delay) becomes large, such that jobs need to be processed quickly. Hence, in the absence of a processor sleep mode, we propose an optimization problem with much lower complexity.

Optimization Problem 2: Minimize a Fixed Power Level given Bounds on the Fraction of Dropped Jobs

$$\begin{aligned}&\min_{k} P_k \\ &s.t.\ \Pr\{D_{i,k} > T_i\} < \varepsilon_i.\end{aligned} \tag{18}$$

Since each power level has I bounds to compute, where I is the number of job classes, the complexity of determining the optimal power level is $O(I \log K)$, where K is the number of

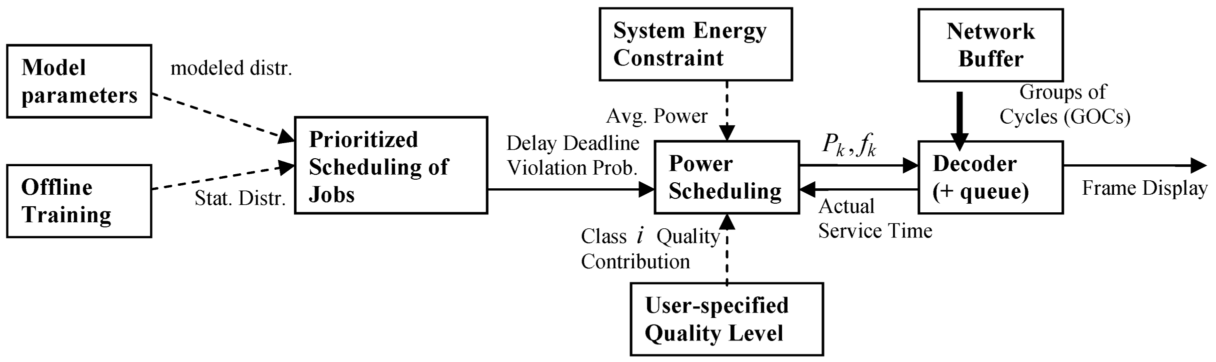


Fig. 6. Queuing model for priority scheduling based DVS.

processor power levels. Due to the strictly monotonically decreasing value of $\Pr\{D_{i,k} > T_i\}$ with respect to K, the probability of missing deadlines is well-ordered with respect to the power levels. Hence, a bisection algorithm can be used to determine the minimum power level such that all of the bounds are satisfied.

We proposed two variations of an adaptive algorithm that performs power adaptation to achieve energy savings while meeting deadlines with high probability. Based on control parameter α, Algorithm 1 below adjusts its power such that the decoding deadlines for all classes will be met with high probability. Hence, it solves (1) after processing each job. Algorithm 2, on the other hand, adjusts the power based only on the class about to be processed, such that that particular class i will meet its decoding deadline with probability $1 - \varepsilon_i$. The complexity for Algorithm 2 per job is thus reduced to $O(\log K)$. Hence, both algorithms have complexity that does not grow with the total number of released jobs, while laEDF has complexity $O(\#\text{ of released jobs} \cdot \log K)$ due its consideration of all job deadlines.

```
Algorithm 1 Adjusting power for all classes
based on service time overshoot or undershoot

1. Solve problem 2.
2. While jobs are available,
3.  Set time $t$ to 0 for each (soft) arrival point.
4.  If job $i$ finishes at time $t = \tau + \Delta\tau$
5.    Change delay bounds for all classes $j$ to $\Pr\{D_{j,k} > T_j - \alpha\Delta\tau\} < \varepsilon_j$, $0 \le \alpha \le 1$
6.    Solve problem 2 under new constraints.
7.  end
8. end
```

```
Algorithm 2 Adjusting power per class based on
service time overshoot or undershoot

1. Solve problem 2. Set $P_{k_1} = P_{k^*}$ for job 1, where $P_{k^*}$ is solution to problem 2.
2. While jobs are available,
3.  Set time $t$ to 0 for each (soft) arrival point.
4.  If job $i$ finishes at time $t = \tau + \Delta\tau$
5.    Change delay bound for class $i+1$ to $\Pr\{D_{i+1,k_i} > T_{i+1} - \alpha\Delta\tau\} > \varepsilon_i$, $0 \le \alpha \le 1$
6.    Find $P_{k_{i+1}}$ by solving problem 2 for class $i+1$ with only the $i+1th$ delay constraint.
7.  end
8. end
```

IV. Priority Scheduling Based DVS

In this section, we introduce the concept of a joint DVS and priority-based job scheduling algorithm (Fig. 6). By dropping less important jobs, we can gracefully degrade the quality by operating at lower power levels. We will show examples of this graceful degradation in our results section.

A. *Incoming Traffic Model and Service Model*

Unlike the deadline-driven decomposition of jobs, we consider jobs that are decomposed based on their contribution to quality (Fig. 7). Jobs are then organized into priority classes, where a job that contributes more to quality belongs to a higher priority class. Consider again a buffer that streams jobs to the decoder. By varying the way the bits are streamed to the decoder, we can derive a model where ED complexity arrives in groups of cycles (GOCs) of fixed size according to a mixed arrival process. Because ED complexity is closely related to the arrival rate of bits, ED GOCs can be seen as a complexity representation of packets. Below, based on some simplifying assumptions, we will show that ED complexity can be modeled by GOCs that arrive according to a Poisson arrival process plus a general arrival process.

Proposition 1: Let us assume the following.

1) Jobs of class i arrive in periodic time intervals of size τ_i.
2) The bit-rate r_i per job of each class i is quasi-constant.
3) The shifted and scaled Poisson distribution for ED complexity per frame holds.
4) Different temporal level transform frames corresponding to the same job have independent ED complexity statistics (as a result of motion compensation).
5) The buffer can feed the decoder with bits that arrive according to an arbitrary distribution, as long as the total number of bits that arrive within time τ_i is r_i.

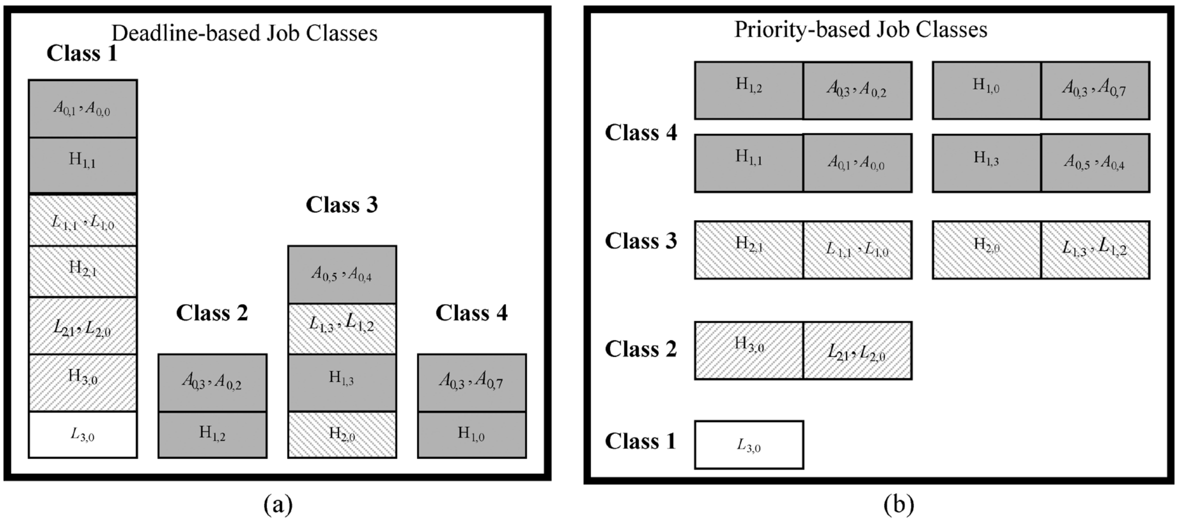


Fig. 7. (a) Job decomposition based on deadlines for 3 level MCTF. (b) An example of job decomposition based on priority classes. Connected frames are associated with the same job.

Then ED complexity can be modeled by a Poisson arrival process plus a general arrival process.

Proof: It is a well-known fact that a Poisson arrival process with i.i.d. exponential interarrival times $a(t) = \lambda_i e^{-\lambda_i t}$ can be decomposed into a doubly stochastic model based on a Poisson distributed number of arrivals within a fixed time period τ, and a uniform distribution of arrivals within that period [37], i.e.,

$$N_A \sim p_A(n) = \frac{(\lambda_i \tau_i)^n e^{-\lambda_i \tau_i}}{n!}, \quad n \geq 0 \tag{19}$$

$$\bar{U}_{N_A} = (u_{1,N_A}, u_{2,N_A}, \ldots, u_{N_A,N_A}) \sim f\left(\bar{U}_{N_A}\right) = \frac{1}{\tau^{N_A}}$$
$$0 \leq u_{N,j} < \tau, \quad j = 1, \ldots N \tag{20}$$

where λ is the arrival rate of the process, N is the random number of arrivals within the period, and $\bar{U}_N$ is the uniform distributed vector over an N_A-dimensional hypercube of all combinations of possible arrival times. (Note that components of $\bar{U}_{N_A}$ do not necessarily have to arrive in order.)

Now, consider the case where ED complexity for a frame follows a Poisson distribution with mean ν_{fr}. The buffer, which can arbitrarily stream bits that it contains for a given job, will stream in such a way that GOCs of size 1 (cycle) arrive at rate ν_{fr}/τ_i uniformly distributed in time interval $[0, \tau_i]$. Hence, the ED complexity for the frame follows a Poisson arrival process. Since wavelet transformed frames are often independent, we may assume the ED complexity associated with each frame in a job to be independent. The sum of independent Poisson arrival processes is another Poisson arrival process, hence the ED complexity per each job, which may include entropy decoding several frames, is a Poisson process. Denote the average ED complexity per job of class i as ν_i. Based on the above construction per frame, the buffer now streams bits in a manner such that, based on the ED complexity associated with each bit, the ED complexity is "streamed" as a Poisson process in GOCs of size 1 (cycle) with arrival rate ν_i/τ_i with total job complexity

$$C_{i,\text{ED}} \sim p_{i,\text{ED}}(n) = \frac{\nu_i^n e^{-\nu_i}}{n!}, \quad n \geq 0. \tag{21}$$

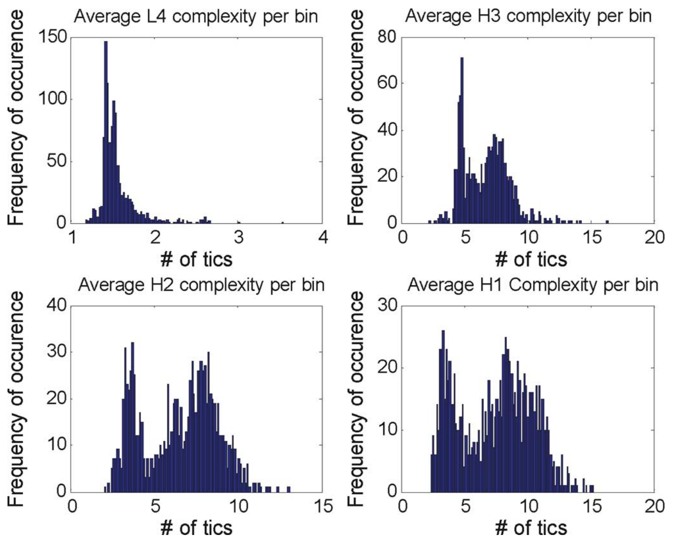


Fig. 8. Example of total complexity per arriving ED GOC for various frames in a 4 temporal level MCTF GOP. The statistics are averaged over several sequences.

Now consider $C_{i,\text{ED}} = a_i \hat{C}_{i,\text{ED}} + b_i$, where $\hat{C}_{i,\text{ED}}$ follows the distribution in (21). First, $a_i \hat{C}_{i,\text{ED}}$ can be modeled as a Poisson process as above, but with GOCs of size a_i. Second, we can model the complexity b_i as arriving across an independent complexity stream, where exactly b_i/a_i GOCs of size a_i arrive within time τ_i. Hence, the total arrival process is a mixture of Poisson and a general arrival process. ■

For the remainder of this section, we simplify the model for ED complexity to be a pure Poisson arrival process. We use $\lambda_1, \lambda_2, \ldots, \lambda_I$ to denote the quantized ED arrival rate for priority classes $1, \ldots, I$, and $\lambda = \sum_{i=1}^{I} \lambda_i$ is the total job arrival rate.

B. Proposed DVS Service Policy and Model

Based on the decomposition of jobs into ED GOCs in Section IV-A, we propose a DVS system that uses priority scheduling to process the incoming GOCs. We model the service of GOCs as a nonpreemptive $M/G/1$ priority queuing system, where G corresponds to the service time distribution of each GOC (example distributions are shown in Fig. 8). In other words, whenever the system finishes servicing a GOC, it will then process the highest priority GOC waiting in the queue at the time. However, while a GOC is being processed, a GOC of higher priority can not interrupt the ongoing service (i.e., nonpreemptive). Based on priority scheduling, the system will ensure that even if not all jobs can be processed before the display deadline, the higher priority jobs will be processed first, so that they are more likely to satisfy their deadline constraints. Effectively, a lower quality video can be streamed by decoding (in time) only jobs in higher priority classes without having to process jobs from every priority class. This creates a quality and energy tradeoff, as we can lower the average processor power to create a video of lower quality. To model the service rate per GOC, we divided the total complexity (in tics) associated with the decoding of each frame by the complexity of entropy decoding.

C. Delay and Idle Time Analysis

Let $D_{i,k}$ be the delay of processing a GOC of class i, and define $\Pr\{D_{i,k} > T_i\}$ to be the probability that a GOC arriving at time t can not be processed before deadline $t + T_i$. Note that in reality, all GOCs of the same job have the same hard deadline regardless of their arrival times t, so the delay bound T_i would not be fixed for every GOC of a job. However, considering that GOCs need to be processed in FIFO order to complete the job, the deadlines for the first GOCs in the job may be set earlier to accommodate the processing time delay induced on later GOCs. For the purpose of analysis, we approximate the delays tolerated by all GOCs within a class to be approximately equal.

In order to determine the probability of violating the delay deadline for a non-preemptive priority queuing system, we first define the load on the system induced by priority class i with service time $S_{i,k}$ as

$$\rho_{i,k} = \lambda_i E[S_{i,k}]. \tag{22}$$

Let $\sigma_{i,k} = \sum_{j=1}^{i} \rho_{j,k}$ be the total load of traffic coming from priority classes 1 to i, and let $\mu_{i,k}$ be the average service rate for a class i job in processor operating mode k. The average waiting time in the queue for priority class i GOCs can then be expressed as [39]

$$\mathrm{E}[W_{i,k}] = \frac{\sum_{j=1}^{I} \frac{\rho_{j,k}}{\mu_{j,k}}}{2(1-\sigma_{i-1,k})(1-\sigma_{i,k})}. \tag{23}$$

From the average waiting time, we can obtain an approximation for the probability that the waiting time exceeds some time t. We use the waiting time tail approximation to estimate the tail of the delay

$$\begin{aligned}\Pr\{D_{i,k} > T_i\} &= \Pr\{W_{i,k} + S_{i,k} > T_i\} \\ &\approx \rho_k \exp\left(-\frac{\rho_k T_i}{\mathrm{E}[W_{i,k}] + E[S_{i,k}]}\right).\end{aligned} \tag{24}$$

The fraction of idle time in an $M/G/1$ queuing system is the time average probability that the system is empty

$$p_{0,k} = 1 - \rho_k. \tag{25}$$

D. Priority Scheduling Optimization Problems and Algorithms

In this section, we formulate and analyze a number of optimization problems based on probabilistic delay constraints. We begin with a simple optimization problem, where a processor continues running an idle processing thread even if there are no jobs in system.

Optimization Problem 3: Minimize the Average Active Power given an Average Video Quality

$$\begin{aligned}&\min_{\alpha=(\alpha_1,\ldots,\alpha_K)} \sum_{k=1}^{K} \alpha_k P_k \\ &s.t. \sum_{k=1}^{K} \alpha_k Q_k \geq Q_{\text{avg}} \\ &\sum_{k=1}^{K} \alpha_k = 1\end{aligned} \tag{26}$$

where

$$Q_k = \sum_{i=1}^{I} \frac{\lambda_i}{\lambda} \Delta_i \Pr\{D_{i,k} \leq t_i\} \tag{27}$$

is the average quality of the decoded sequence at power level P_k. Here, α is a vector with components that are the fraction of time the processor is set to operate at power level P_k, and Δ_i is the quality slope parameter for priority i GOCs (i.e., the average quality contributed to video by a priority i GOC.) as introduced in [32]. Note that λ_i/λ is the fraction of GOCs of priority i received from the bitstream. Thus, the first constraint requires that the average quality of the video is at least Q_{avg}. This problem turns out to be a linear programming problem, since P_k and Q_k are constants. We can thus solve this via the simplex method. However, an even simpler closed-form solution exists if we explicitly consider the properties of power with respect to quality.

Proposition 2: If quality is a concave increasing function of ED complexity, and there are a finite number of power levels, the optimal solution to Optimization Problem 3 is to run the processor always at a single power level, or perform time sharing between two adjacent power levels.

Proof: Let Q be a discrete random variable which takes on quality levels with probability. Power is a convex function of frequency [28]. Likewise, complexity (and, thus, the average processor frequency per unit time) is a convex function of quality, which can been shown theoretically [25], [26] given a concave PSNR curve with respect to rate. Hence, the power is a convex function of the required average quality. From [38],

TABLE III
FREQUENCY CHANGES FOR A COMBINED SEQUENCE WITH ABOUT 8600 JOBS

DVS Algorithm	laEDF DVS	Algorithm 1	Algorithm 2
# Freq. Changes	8365	2124	1658

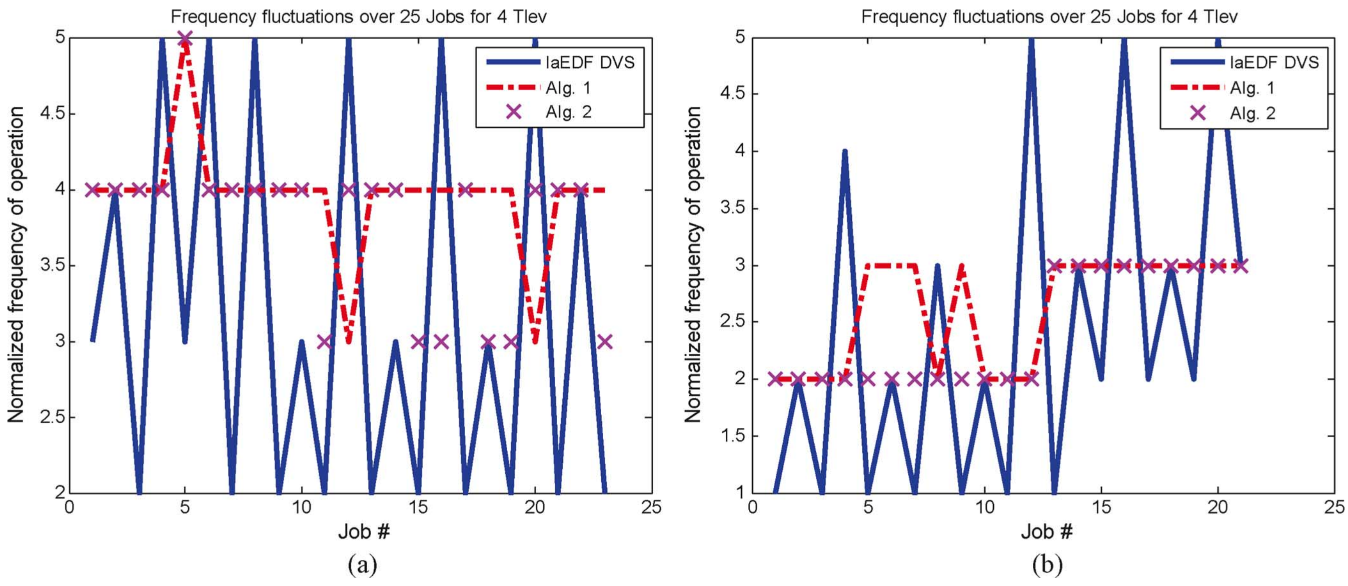


Fig. 9. Frequency and power fluctuations over 20 jobs of *Coastguard* for (a)–(b) 2 temporal levels temporal and (c)–(d) 4 temporal levels.

it is shown that for a convex quality to power function , the distribution that minimizes the expected value of the function $E[P(\widehat{Q})]$ is to choose the with probability 1 if , or else

$$\widehat{Q} = \begin{cases} Q_{k^*} & w.p.\ \frac{Q_{\text{avg}}-Q_{k^*}}{Q_{k^*+1}-Q_{k^*}} \\ Q_{k^*+1} & w.p.\ \frac{Q_{k^*+1}-Q_{\text{avg}}}{Q_{k^*+1}-Q_{k^*}} \end{cases} \tag{28}$$

where $Q_{k^*} < Q_{\text{avg}} < Q_{k^*+1}$. $\widehat{Q}$ then minimizes $E[P(\widehat{Q})]$, which gives us the solutions α_k to Optimization Problem 3. ■

If we now consider the case where the processor may shut down during idle times and expend essentially zero energy, we have a different optimization problem.

Optimization Problem 4: Minimize the Average Power given an Average Video Quality

$$\min \sum_{k=1}^{K} \alpha_k \rho_k P_k$$
$$s.t. \sum_{k=1}^{K} \alpha_k Q_k \geq Q_{\text{avg}}, \sum_{k=1}^{K} \alpha_k = 1 \tag{29}$$

This problem is the same as Optimization Problem 3 but under a different objective function which is not necessarily convex. Again, we assume that the static power consumption can be neglected due to very high active power consumption. Hence, the optimal mode of operation should keep the system nonempty with high probability, such that the processor power should run at a nearly constant power level [28]. We propose a simplified problem that can be solved with complexity $O(I \cdot \log K)$ and can be used by a DVS algorithm to reactively adapt the power level based on a minimum desired average quality.

Optimization Problem 5: Minimize a Fixed Power given an Average Quality

$$\min P_k$$
$$s.t.\ Q_k \geq Q_{\text{avg}}. \tag{30}$$

We propose several simple priority scheduling and power scheduling algorithms for DVS. The first algorithm chooses a constant power based on the arrival rate and service time statistics by solving Optimization Problem 5 with various levels of Q_{avg}. The second algorithm is the same as the first, but periodically purges the queue of expired jobs, thereby reducing the average waiting time for different classes. Finally, we present a combined quality-aware priority and look-ahead algorithm (Algorithm 3), which temporarily increases the power whenever important jobs are about to expire. Whenever a job in a class i is within δ s of being expired, the system will increase the processor power according to the job's priority by some $\Upsilon(i)$, thereby increasing the chance of that job being decoded in time.

```
Algorithm 3 Priority scheduling with last
second power increase

1. Solve problem 5 for $Q_{\text{avg}}$ to obtain $P_{init}$.
2. While jobs are available,
3.  For the highest priority class $i$, such that
the deadline of a job in class $i$ will expire in
less than $\delta$ time
4.     Set $P = P_{init} + \Upsilon(i)$.
5.  end
6.  Process highest priority job in FIFO
order. Record service time $s$
7.  Subtract deadline of all other jobs by $s$.
```

TABLE IV
COMPARISONS OF ENERGY CONSUMPTION, POWER LEVELS USED, AND DEADLINES MISSED FOR VARIOUS DVS ALGORITHMS ON THE COMBINED SEQUENCE. THERE ARE A TOTAL OF FIVE POWER LEVELS. THE LEFT NUMBER IS FOR TWO TEMPORAL LEVELS, AND THE RIGHT NUMBER FOR FOUR TEMPORAL LEVELS

Algorithm used:	**Energy consumed:**		**Power/Frequency Levels used:**		**Deadline Miss %:**	
# Temporal Levels	***2***	***4***	***2***	***4***	***2***	***4***
No DVS(const. power)	3.45E	3.87E	4	4	0.01%	0.09%
laEDF DVS	2.65E	3.58E	1-5	1-5	0%	0.51%
Algorithm 1 ($a = 0.5$)	2.38E	3.15E	2-3	2-3	0.02%	0.07%
Algorithm 2 ($a = 0.5$)	2.38E	3.12E	2-3	2-4	0.02%	0.07%

TABLE V
COMPARISONS OF PERFORMANCES OF VARIOUS PRIORITY SCHEDULING ALGORITHMS IN TERMS OF THE PERCENTAGE OF DEADLINES MISSED FOR VARIOUS PRIORITY CLASSES FOR FOUR TEMPORAL LEVEL DECOMPOSITION (f_0 INDICATES THE MINIMUM PROCESSOR POWER)

Jobs decoded in time (%)	Frequency Levels ®	f_0	$2f_0$	$3f_0$	$4f_0$	$5f_0$
Priority Scheduling	class 1	99.72	100	100	100	100
	class 2	67.33	99.91	100	100	100
	class 3	0	98.48	99.91	100	100
	class 4	0	0	0	99.05	99.95
	class 5	0	0	0	0	0
Priority Scheduling with Queue Purging	class 1	99.62	100	100	100	100
	class 2	68.18	99.91	100	100	100
	class 3	13.54	99.72	100	100	100
	class 4	0	14.32	57.50	99.67	99.95
	class 5	0	0	6.46	26.95	41.28
Priority Scheduling with Last Second Power Increase for 3 Priority Classes	class 1	99.91	100	100	100	100
	class 2	91.29	99.91	100	100	100
	class 3	31.91	99.43	100	100	100
	class 4	0	14.02	58.43	99.67	99.95
	class 5	0	0	6.63	26.95	41.28

```
8. If deadline of a job $j$ is less than 0,
purge job $j$.
9. end
```

V. EXPERIMENTATION AND RESULTS

In this section, we compare the performance of deadline-driven Algorithm 1 and Algorithm 2 with laEDF DVS. We then compare the performances of different priority scheduling algorithms and discuss the quality-energy adaptation points achieved by Algorithm 3.

A. Deadline-Driven DVS Results

Based on the assumption of zero passive power, we compare our algorithms with no DVS and laEDF DVS below using five equally spaced frequency levels. For our simulations, we combined 11 video sequences of 16 GOPs each (e.g., *Coastguard*, *Foreman*, *Harbor*, *Mobile*, *Silent*, *Stefan*, and several others) into a long sequence, which was then decoded. We set the hard deadlines for Algorithm 1 and Algorithm 2 to be 8 frames after the (soft) periodic arrivals, and we collected data for both 2 temporal level and 4 temporal level MCTF sequences. When choosing the steady state deadline violation probability constraints ε_i for the algorithms, we noticed that for $\varepsilon_i \approx 1$, the average delay tended to be large, such that not only were deadlines frequently missed, but the power level per job also varied significantly due to trying to meet imminent deadlines. However, for small ε_i, the power setting per job was on average higher, since there was more idle time between jobs. Based on experimental results where the probability of deadline violation for every class of jobs is be equal, we found the optimal setting to be $\varepsilon_i = 0.75$ for all classes i.

In Table III, the combined sequence is decoded using a 4-temporal level MCTF GOP structure, and the number of processor frequency changes are compared between different algorithms. Interestingly, Algorithm 2 had the fewest frequency changes, but both queuing-model-driven algorithms switched operating levels about 20%–25% as often as laEDF DVS. Hence, our algorithms greatly reduces the switching overhead. To illustrate this effect, Fig. 9 shows an example of (normalized) frequency fluctuations induced by the various DVS policies for 25 consecutive jobs of the *Coastguard* and *Stefan* sequences, encoded with 4 temporal level MCTF. Note that frequency levels vary dramatically for laEDF DVS due to the highly varying job complexities per class.

In Table IV, we compared the average energy, power fluctuations, and deadline misses for the combined sequence using various algorithms. Note that our queuing-based algorithms had nearly the same performance in spite of different power schedules, and when compared with the laEDF DVS, provided approximately 10% energy savings for 2 temporal level MCTF and 15% energy savings for 4 temporal level MCTF.

B. Priority Scheduling DVS Implementation and Results

For the priority scheduling approach, we decomposed jobs from the MCTF GOP structure in the same way shown in Fig. 5. We note that while there are other ways to prioritize jobs based on different classes for single frames, for resolution levels within frames, or even for subbands within resolution levels in order to achieve results with finer granularity, the

TABLE VI
COMPARISONS OF PERFORMANCES OF VARIOUS PRIORITY SCHEDULING ALGORITHMS IN TERMS OF THE PERCENTAGE OF DEADLINES MISSED FOR VARIOUS PRIORITY CLASSES FOR 2 TEMPORAL LEVEL MCTF DECOMPOSITION

Jobs decoded in time (%)	Frequency Levels ®	f_0	$2f_0$	$3f_0$	$4f_0$	$5f_0$
Priority Scheduling	class 1	98.60	100	100	100	100
	class 2	0	0	100	100	100
	class 3	0	0	0	0	0
Priority Scheduling with Queue Purging	class 1	100	100	100	100	100
	class 2	0	38.07	100	100	100
	class 3	0	0	6.32	28.86	67.19
Priority Scheduling w/ Last Second Power Increase for 2 classes	class 1	100	100	100	100	100
	class 2	7.36	78.95	100	100	100
	class 3	0	0	6.32	28.86	67.19

TABLE VII
COMPARISONS OF AVERAGE ENERGY CONSUMPTION AND QUALITY LEVELS FOR ALGORITHM 2 AND QUALITY-ENERGY ADAPTATION POINTS OF ALGORITHM 3 FOR THE *COASTGUARD* AND *STEFAN* SEQUENCES DECODED A BIT RATE 768 kb/s. THE AVERAGE FRAME RATES (FRAMES PER SECOND) OVER ALL GOPS ARE GIVEN FOR DIFFERENT ADAPTATION POINTS FOR ALGORITHM 3

	Frame rate (fps):		**Energy consumed:**		**PSNR (dB):**	
Algorithm	***Coastguard***	***Stefan***	***Coastguard***	***Stefan***	***Coastguard***	***Stefan***
2	30.00	30.00	2.63E	2.41E	33.24	27.35
3	26.48	23.67	2.15E	2.15E	32.98	27.01
3	20.04	18.05	1.26E	1.25E	32.51	26.70
3	16.17	15.23	0.65E	0.65E	32.23	26.48
3	14.53	10.08	0.28E	0.29E	32.05	25.94
3	8.09	7.27	0.09E	0.09E	30.68	25.55
3	5.04	3.63	0.02E	0.03E	29.44	24.62

priority classification method depicted in Fig. 5 was sufficient for our priority-driven DVS implementation.

Based on various average power levels for the processor, we compared the probability of dropping jobs of different classes based on the strict priority scheduling policy, the periodic queue purging priority policy, and our DVS strategy in Algorithm 3. Table V includes the results from the combined sequence encoded by 4 temporal level MCTF, and Table VI includes the same results for 2 level MCTF based on some lowest operating frequency f_0. For Algorithm 3, we used $\Upsilon(1) = 1.5\Upsilon(2) = 3\Upsilon(3) = 3f_0$ for 4 temporal level MCTF with five priority classes, and $\Upsilon(1) = 2\Upsilon(2) = 2f_0$ for 2 temporal level MCTF with 3 priority classes. While Algorithm 3 may expend slightly more power than pure priority queuing due to speeding up when jobs are about to expire, it performs better than pure priority queuing due to reactively rushing jobs through at the last minute. In the case of job classes being frames or groups of frames, the effect of dropping different priority classes is the same as reducing the frame rate of the corresponding GOP. Finally, Table VII shows how different power levels correspond to different frame rates, energies, and quality levels. Notice that as long as the frame rate is sufficiently high (e.g., 10 fps), there is only a loss in quality of less than 1.5 dB when the power is scaled down to 10%, which demonstrates that Algorithm 3 achieves high-scalability in terms of quality and power tradeoffs.

VI. CONCLUSION

Current multimedia compression algorithms and standards provide only very coarse levels of complexity, thereby neglecting the vast resource diversity and heterogeneity of state-of-the-art systems. Also, current systems lack good complexity models for resource management, and, hence, statistics must be collected and updated frequently online in order to reactively adapt to time-varying source and coding structures. To improve the implementation of multimedia applications on voltage/power configurable systems, we constructed a (fine) granular complexity model that can be adapted in real-time by a low overhead transmission from the encoder/server. Based on this model, we proposed a queuing-driven DVS approach for a video decoding system, which achieves significant energy savings compared to conventional DVS. Finally, we proposed an adaptive architecture combining both power and job scheduling to obtain energy-quality tradeoffs. Our results indicated that the priority-scheduling based DVS algorithms can save a significant amount of energy with only a small reduction to the quality level. Future work includes the consideration of high static power, and of several multimedia tasks that are running simultaneously on the same system.

REFERENCES

[1] Intel Inc., Intel XScale Technology [Online]. Available: http://www.intel.com/design/intelxscale
[2] AMD Inc., AMD PowerNow!TM Technology Platform Design Guide for Embedded Processors [Online]. Available: http://www.amd.com/epd/processors
[3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, MA: Kluwer Academic, 1997.
[4] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th ACM Symp. Operat. Syst.*, 2001.
[5] V. Raghunathan, C. Pereira, M. Srivastava, and R. Gupta, "Energy aware wireless systems with adaptive power-fidelity tradeoffs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Feb. 2005.
[6] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets, "GRACE: Cross-layer adaptation for multimedia quality and battery energy," *IEEE Trans. Mobile Comput.*, to be published.
[7] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," in *Proc. 11th Int. Conf. Comp. Architect.*, 2004.

[8] D. H. Albonesi, R. Balasubramonian, S. Dropsho, S. Dwarkadas, E. G. Friedman, M. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster, "Adaptive processing: Dynamically tuning processor resources for energy efficiency," *IEEE Comput.*, Dec. 2003.

[9] S. Irani, G. Singh, S. K. Shukla, and R. K. Gupta, "An overview of the competitive and adversarial approaches to designing dynamic power management strategies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 12, pp. 1349–1362, Dec. 2005.

[10] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, "Frame-based dynamic voltage and frequency scaling for a MPEG decoder," in *ICCAD*, 2002, pp. 732–737.

[11] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 5, May 2004.

[12] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 1998, pp. 76–81.

[13] W. Yuan and K. Nahrstedt, "Practical voltage scaling for mobile multimedia devices," in *Proc. 12th Ann. ACM Int. Conf. Multimedia*, New York, Oct. 10–16, 2004.

[14] A. Maxiaguine, S. Chakraborty, and L. Thiele, "DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs," in *Proc. 3rd IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and Syst. Synth.*, 2005.

[15] Z. Ren, B. Krogh, and R. Marculescu, "Hierarchical adaptive dynamic power management," *IEEE Trans. Comput.*, vol. 54, no. 4, Apr. 2005.

[16] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. de Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proc. Design Automat. Conf.*, 2001.

[17] J. Liu, W. Shih, K. Lin, R. Bettati, and J. Chung, "Imprecise computations," *Proc. IEEE*, 1994.

[18] M. Mesarina and Y. Turner, "Reduced energy decoding of MPEG streams," *Multimedia Syst.*, 2003.

[19] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Process.*, 1993.

[20] P. Schelkens, A. Munteanu, J. Barbarien, M. Galca, X. Giro-Nieto, and J. Cornelis, "Wavelet coding of volumetric medical datasets," *IEEE Trans. Med. Imag.*, vol. 22, no. 3, Mar. 2003.

[21] O. Boxma and W. Groenendijk, "Waiting times in discrete-time cyclic-service systems," *IEEE Trans. Commun.*, vol. 36, no. 2, Feb. 1988.

[22] T. J. Ott, "Simple inequalities for the D/G/1 queue," *Operat. Res.*, 1987, INFORMS.

[23] L. D. Servi, "D/G/1 queues with vacations," *Operat. Res.*, 1986, INFORMS.

[24] H. Zhang and S. C. Graves, "Cyclic scheduling in a stochastic environment," *Operat. Res.*, 1997, INFORMS.

[25] D. M. Sow and A. Eleftheriadis, "Complexity distortion theory," *IEEE Trans. Inf. Theory*, vol. 49, no. 3, pp. 604–608, Mar. 2003.

[26] B. Foo, Y. Andreopoulos, and M. van der Schaar, "Analytical rate-distortion-complexity modeling of wavelet-based video coders," *IEEE Trans. Signal Process.*, accepted for publication.

[27] Y. Lim and J. Kobza, "Analysis of a delay-dependent priority discipline in a multi-class traffic packet switching node," in *Proc. IEEE INFOCOM*, Apr. 1988, pp. 889–898.

[28] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. ACM ISLPED*, 1998, pp. 197–202.

[29] M. van der Schaar and Y. Andreopoulos, "Rate-distortion-complexity modeling for network and receiver aware adaptation," *IEEE Trans. Multimedia*, vol. 7, no. 3, pp. 471–479, Jun. 2005.

[30] S. Regunathan, P. A. Chou, and J. Ribas-Corbera, "A generalized video complexity verifier for flexible decoding," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2003, vol. 3, pp. 289–292.

[31] Y. Andreopoulos and M. van der Schaar, "Adaptive linear prediction for resource estimation of video decoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 6, pp. 751–764, Jun. 2007.

[32] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Process. Mag.*, vol. 15, no. 6, pp. 23–50, Nov. 1998.

[33] J. Ohm, M. van der Schaar, and J. Woods, "Interframe wavelet coding-motion picture representation for universal scalability," *Signal Process.: Image Commun.*, vol. 19, pp. 877–908, 2004.

[34] F. Fitzek and M. Reisslein, "MPEG-4 and H.263 video traces for network performance evaluation," *IEEE Network*, Nov.–Dec. 2001.

[35] T. Jiang, C. K. Tham, and C. C. Ko, "An approximation for waiting time tail probabilities in multiclass systems," *IEEE Commun. Lett.*, vol. 5, no. 4, pp. 175–177, Apr. 2001.

[36] J. Abate, G. L. Choudhury, and W. Whitt, "Exponential approximations for tail probabilities in queues I: Waiting times," *Operat. Res.*, vol. 43, no. 5, pp. 885–901, 1995.

[37] R. G. Gallager, *Discrete Stochastic Processes*. Dordrecht, Germany: Kluwer, 1996.

[38] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

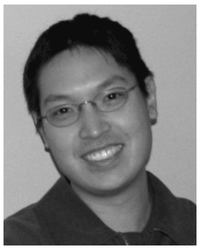
[39] D. Gross and C. Harris, *Fundamentals of Queueing Theory*. New York: Wiley-Intersci., 1997.

[40] Y. Yoo, A. Ortega, and B. Yu, "Image subband coding using progressive classification and adaptive quantization," *IEEE Trans. Image Process.*, pp. 1702–1715, Dec. 1999.

[41] M. Ravasi, M. Mattavelli, P. Schumacher, and R. Turney, "High-level algorithmic complexity analysis for the implementation of a motion-JPEG2000 encoder," in *PATMOS*, 2003, pp. 440–450.

[42] M. Mattavelli and S. Brunetton, "Implementing real-time video decoding on multimedia processors by complexity prediction techniques," *IEEE Trans. Consum. Electron.* vol. 44, no. 3, pp. 760–767, Aug. 1998 [Online]. Available: http://www.intel.com/design/intelxscale, Intel Inc. "Intel XScale Technology"

[43] Q. Wu, P. Juang, M. Martonosi, and D. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *Proc. 11th Int. Conf. Architect. Support for Programm. Languages Operat. Syst. (ASPLOS)*, 2004.



Brian Foo received the B.S. degree from the University of California, Berkeley, in electrical engineering and computer sciences in 2003. He received the M.S. degree in electrical engineering from the University of California, Los Angeles (UCLA), in 2004.

He is pursuing the Ph.D. degree under Prof. van der Schaar at UCLA.

Mihaela van der Schaar (SM'04) is currently an Assistant Professor with the Electrical Engineering Department, University of California, Los Angeles (UCLA). Since 1999, she was an active participant to the ISO MPEG standard to which she made more than 50 contributions. She holds 28 granted U.S. patents.

Dr. van der Schaar received three ISO recognition awards, the NSF CAREER Award in 2004, an IBM Faculty Award in 2005, and an Okawa Foundation Award in 2006. She has also received the Best IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY Paper Award in 2005 and Most Cited Paper Award from EURASIP *Journal Signal Processing: Image Communication* between 2004–2006. She is currently an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, IEEE SIGNAL PROCESSING LETTERS, and the IEEE *Signal Processing e-Newsletter*. She is also the editor (with P. Chou) of the book *Multimedia over IP and Wireless Networks: Compression, Networking, and Systems*.