Coarse-Grained Learning-Based Dynamic Voltage Frequency Scaling for Video Decoding

Jia Guo and Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles
Email: {jia, miodrag}@cs.ucla.edu

***Abstract*—Dynamic Voltage and Frequency Scaling (DVFS) is widely used in today's mobile devices. Commonly adopted OS level DVFS policies usually operate with high sampling and adjustment frequency in order to compensate for the lack of information from applications. In this paper, we consider the application of video decoding and propose a method that enables DVFS at a coarse time granularity by taking advantage of application level semantics. Using machine learning, we build a prediction model for the choice of CPU frequency based on the estimated workload and the progress of execution. We show that with our method, the system is able to perform DVFS with a frequency of only around 1 Hz, abiding by the practical constraints imposed by existing operating systems, but still saves an average of 40.1% in energy consumption compared with the execution at nominal speed. Further, we argue that the law of diminishing returns applies if we increase the frequency at which DVFS operates, and thus it may be costly and impractical to perform application level DVFS at a very fine time granularity.**

I. Introduction

To prolong the lifetime of battery operated mobile devices, dynamic voltage and frequency scaling (DVFS) has long been regarded as a highly effective technique[1][2]. In practice, OS and lower level DVFS policies are widely used in today's mobile systems, with the representative example of Android's CPUFreq governors inherited mostly from Linux [3][4].

For example, the Nexus 4 Android smartphone supports the following 5 governors: `powersave`, `performance`, `ondemand`, `interactive` and `userspace`, with `ondemand` governor being the default option. Aside from the `userspace` governor, which is merely an interface for the user to change the CPU frequency, the rest are controlled by the kernel. The `powersave` governor simply enforces the use of the lowest frequency. Similarly, the `performance` governor simply configures the CPU to use the highest frequency. The `ondemand` governor samples the CPU usage every tens of milliseconds (in Nexus 4's case, 50 milliseconds) and adjusts the voltage and frequency level according to the usage statistics. The `interactive` is a improved version of `ondemand` governor tuned for interactive workloads. Since the operating system is unaware of the application level workload, the only way to manage energy CPU frequency is to "guess" the future workload based on the sampling results. On the one hand, if the sampling rate is too low, the CPU might not adjust to change in workload fast enough. On the other hand, if we have too high a sampling rate, it will incur a huge overhead considering the *transition latency* in CPU frequency switching. Thus, the sampling rate mentioned above must have been carefully tuned by the engineers to maximize the effectiveness of the DVFS policy. As a result, with those speculative, but frequent decisions, our operating system is able to correct itself on the brink of mistakes, providing us with just about the reasonable level of quality of service.

In recent years, as smartphones become more prevalent, new problems are exposed and more researchers start to look beyond the operating system to higher levels. From the revelation of energy bugs found on the application level[5] to the proposal of application sensitivity to in order to save energy while preserving user experience[6], application and even user-level semantics provide an increasingly important role in improving system energy efficiency. It is also our firm belief that by leveraging the information available at higher levels, the energy management decisions will be much more targeted and precise. For many applications, making DVFS decisions at the application level promises to yield significant improvement in energy consumption.

However, there are practical constraints for managing DVFS in application level. The `userspace` governor allows setting CPU frequency through an interface in the file system often referred to as the in-memory sysfs[7][8]. Yet that interface is not as efficient as we expect it to be. Researchers have shown that it takes an average of 1.7 milliseconds for the voltage change to take place on certain embedded platforms due to various operation system level overhead[9]. It means that if a new policy is to be designed to take advantage of application level semantics, frequency adjustments can only happen with an interval much larger than tens of milliseconds. After all, if the `ondemand` governor already samples at intervals of 50 milliseconds, we should be able to sample at even lower rates and achieve reasonable energy savings given application level semantics,

In this paper, we target the DVFS policy for the application of video decoding considering the new constraint. We raise two key questions: can we accurately predict the most efficient voltage and frequency using application level semantics? Can DVFS with a lower rate provide desirable energy savings? In other words, we first want to show that it is possible to predict the choice of CPU frequency with application level information. Further, we aim at demonstrating that, frequency adjustment can happen at intervals of one or several seconds

978-1-5090-0733-2/16/$31.00 ©2016 IEEE

while still maintaining a high level of energy saving.

In our method, we utilize a decoding buffer already exist in most of the decoders, so that whole video segments are stored first in the buffer before being displayed. The use of buffer smooths out the variation in frame-level decoding computation so that the CPU frequency does not have to reflect the change from decoding I-frames to B-frames or P-frame. However, the buffer alone is not sufficient to allow our coarse-grained DVFS to meet real-time system requirements. The key enabler is an accurate prediction model for future CPU frequencies. The intuition behind our prediction model is that both the estimation of the next video segment and the progress of decoding current segment should be considered when we determine the future voltage and frequency. If the system is behind the schedule, voltage and frequency should be increased. On the other hand, if the system is ahead of schedule, the system should lower the voltage and frequency. Based on offline system statistics, including the decoding workload and buffer status, we build prediction model using machine learning. With the model, we are able to achieve near-optimal energy savings online.

The novelties of this paper are as follows. First, to the best of our knowledge, we are the first to propose a machine learning-based prediction model using both workload estimation and decoding progress as features. Second, different from previous works where DVFS for video decoding usually operates on the frame or group-of-pictures (GOP) level [10][11][12], our proposed method considers frequency adjust rate constraints and switches frequency for every video segment of 1 second or longer. We argue that by taking advantage of buffers and our prediction model, even operating at coarse time granularity, DVFS could still produce desired energy savings. Further, we show that there is a law of diminishing return with respect to energy savings, when we use finer and finer time granularity of DVFS. At one point the energy savings stops increasing as we use higher adjustment frequencies. Not only can our per-segment DVFS method be decoupled from the decoding software, enabling effortless implementation, but it can also tolerate with the practical constraints imposed by existing operating systems.

The rest of the paper is organized in the following order. Section IV outlines the model we use and the assumptions we make about the system. Section IV details the method proposed. Section V evaluates the performance of the system. Section II briefly surveys past effort on DVFS for multimedia applications.

II. Related Work

A. DVFS on Mobile Devices

As mobile devices are becoming more and more prevalent, many researchers are identifying and solving practical problems that exist in off-the-shelf products. Carrol *et al.* studied the integration of core offlining and DVFS using the latest smartphone platforms[13]. Kim *et al.* identifies problems in existing DVFS based thermal management schemes, and proposed a new temperature-aware DVFS algorithm [14]. To solve the same problem of thermal management, Das *et al.* proposed a reinforcement learning based algorithm which integrates not only the frequency management but also the core mapping in a real multicore system [15].

B. Energy Management for Video Decoding

Many previous works have been done to explore the optimal strategy of saving energy for video decoding using DVFS. However, none of the papers listed in this subsection considered the practical constraint of the rate of voltage and frequency change in Android. We categorize these efforts into three broad categories: content aware, workload predicting and buffer assisted.

Content Aware. Techniques such as slack reclaiming have been proposed in the past for dynamic power management for real-time system [16]. Several researchers proposed to use the execution time distribution obtained from offline profiling to guide Dynamic Voltage Scaling such that the system meets the expected requirement[17][18][19].

A more straightforward way is to annotate the content. Chung *et al.* proposed a content provider-assisted lower power video decoding system, where the authors assumed the server would provide the per frame worst-case execution time (WCET) [20]. Using machine learning, Hamers *et al.* were able to categorize scenes in videos into classes, and the server would then annotate scenarios before delivering to the client[12]. Ma *et al.* built linear models for each complexity unit, and assumed decoder could use parameters embedded in videos to predict computation [11]. In all these cases, a clear difference from our proposal is that they all require some type of offline knowledge, either the WCET or annotations from servers. In our model, however, we assume the system is completely unaware of the decoding complexity of the video that it is about to decode.

Workload Predicting. Various methods have been proposed to predict decoding workload. One of the most straightforward ideas is to use the time series related techniques. Choi *et al.* are among the first researchers to propose workload prediction for better energy management for video decoding [10]. Specifically, they separated the decoding computation into a frame-dependent part and a frame-independent part, and apply moving or weighted averages to predict frame-level workload of decoding the frame-independent part. We borrow their idea of workload prediction, but we conduct the prediction on video segment level due to newer system constraints. A similar approach was adopted by Kumar *et al.* [21].

Buffer Assisted. Im *et al.* proposed to add buffers in order to fully utilize idle CPUs cycles, although they assumed periodic tasks and derived analytic solutions based on WCET [22]. Lu *et al.* developed a strategy where the system inserts buffers between different stages of application execution in order to enable frequency scaling and smooth execution [23]. Maxiaguine *et al.* addressed the problem of DVFS with constrained buffer by taking into account both the offline workload bounds and the execution history at runtime.

III. Preliminaries

We consider a system where videos exist in the form of a collection of small segments, as in most cases of internet video streaming. We assume that a single CPU core is responsible for decoding the videos. Note that although GPU is present in most of today's smartphones, CPU still undertakes the video decoding workload in many cases. Such observation is validated in a recent energy consumption analysis of popular apps [24]. In our model, the CPU core is capable of performing DVFS, and the voltage and frequency adjustments happen every video segment. After video segments are decoded, they are stored in a decoding buffer. The decoding buffer has a fixed size, and once the buffer is full, the extra work done for decoding video segments will be wasted. After video segments are decoded and stored in the buffer, they will be fetched in order and passed to graphics hardware for displaying. Figure **??** shows an overview of our model.

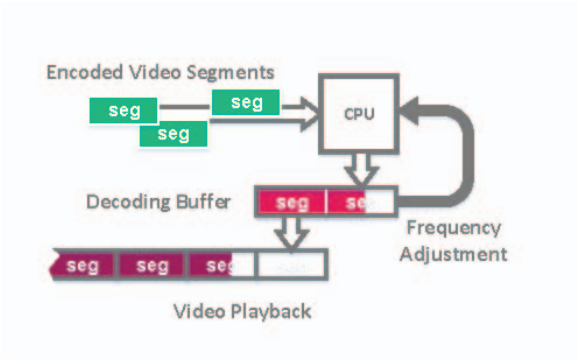

Fig. 1. An overview of the system model.

Consider a video that consists of N video segments, where each segment is of length T seconds. We denote the computational workload (number of CPU cycles) required to decode segment i as C_i. We define a CPU frequency assignment F to be a list of supported CPU frequencies used to decode the corresponding video segments, $F = (f_1, f_2, \ldots, f_N)$, where f_i represents the CPU frequency for decoding the ith video segment. Thus the execution time τ_i of the particular segment i is given by $\tau_i = C_i \cdot f_i$. The resulting energy consumed is denoted as E_i.

In our definition, a *feasible* frequency assignment is defined to be one that meets the following two conditions: a) the decoding buffer does not *underflow*; b) the decoding buffer does not *overflow*. Underflow happens when a video segment finishes playing while the decoding of the next segment has not been completed. Overflow happens when the system decodes too many video segments ahead of time and the video buffer cannot hold the decoded content. The optimal CPU assignment is the feasible assignment that consumes the minimal overall decoding energy $\sum_i E_i$.

IV. Approach

Figure 2 shows the workflow of the proposed method. In the offline phase, we obtain the optimal frequency assignment on the training dataset. Using the statistics generated offline, we train our frequency prediction model that is based on the workload estimation and the progress of decoding (quantified by the slack time). When we test our performance online, we use the workload estimation module to project the computation of the next segment. We use that projection together with the online buffer status to predict next CPU frequency. The following subsection details the steps in this workflow.

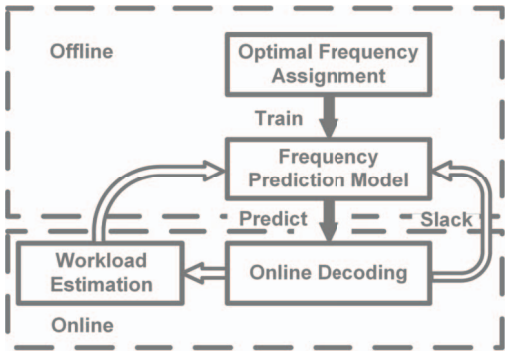

Fig. 2. An overview of our method.

A. Workload Estimation

An important assumption we have about our problem is that we can estimate the decoding complexity of the next segment given the computation it takes to decode the current segment. The assumption is based on an observation we make when we experiment with our video dataset. Figure 3 shows some exemplary computational workload of video decoding, featuring four exemplary videos that are widely used in studies. The y axis shows the decoding workload in number of cycles, and the x axis shows the sequence number of the segments. In this experiment, videos are split into 1-second segments[1]. In our limited number of test videos, the computational workload for decoding neighboring video segments falls in a small range. On average, we observe a 10.2% difference in computation between the current and the next segment. Figure 3 shows 4 exemplary traces. The video `akiyo`, which shows a news anchor, has constantly small workload due to limited inter-frame changes. On the other end of the spectrum is the video `mobile calender` (denoted as `mobile`), featuring a close-range footage of a moving toy train. The video `pamphlet`, despite having relatively large variations in decoding workload, is still not comparable to videos such as `stefan` and `mobile calendar` in terms of decoding complexity. Part of the reason for this consistency, as we mentioned earlier, is because using segment as a unit has smoothed out the variances on the frame level. With that said, we argue that it is feasible to infer the approximate range of, if not accurately predict, the computational workload of decoding future segments in the same video given the history of decoding its past segments.

[1]Unless otherwise indicated, the segment length in our experiment is 1 second throughout the paper

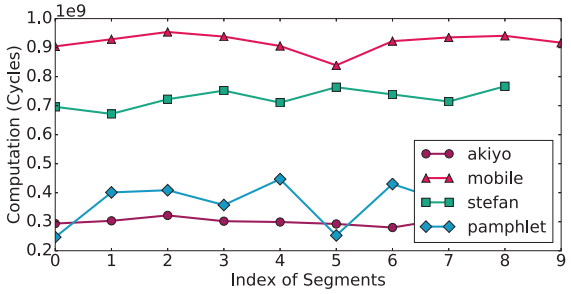

Fig. 3. Examples of video segment decoding workload.

Apparently, we are not the first to analyze the video decoding computation with regard to the temporal relationship among decoding units. In [10], the authors used the weighted average smoothing algorithm to estimate the computational complexity of decoding the next frame

$$C_{i+1} = \alpha \sum_{n=0}^{i} (1-\alpha)^n C_{i-n}$$

However, we show here that such characteristics persist when we zoom out from frame level to segment level. The method mentioned above is well suited for our purposes and we adopted it as our estimation method, although instead of predicting with frames, we predict with video segments.

One of the concerns regarding the workload estimation method is whether our dataset is representative and that if the method will perform well in face of another dataset. Admittedly, our dataset can never be large enough to reflect all of the variability in the multimedia contents on today's internet. Yet the uncertainty can always be addressed by using a larger decoding buffer. More importantly, we by no means rely only on the estimation in predicting the CPU voltage and frequency level for decoding the coming video segment. The progress in decoding the current video segment also plays an equally important role.

B. Offline Frequency Assignment

Recall that our goal for the energy management is to find the best CPU frequency for decoding each video segment. The key task in the offline phase is to search for that optimal solution for a given video, under the condition that the decoding workload for each segment is known in advance. To deal with that problem, we propose an algorithm outlined in Algorithm 1. In short we do an exhaustive search in all the feasible solution space, and use dynamic programming to reduce the amount of search needed to be done. For each feasible frequency assignment F_j to decoding video segments up until segment M, where $F_j = (f_{1j}, f_{2j}, \cdots, f_{Mj})$, we inspect the corresponding tuple that contains the energy and time spent so far $(\sum_{i=1}^{M} E_{ij}, \sum_{i=1}^{M} \tau_{ij})$. Among all the tuples, we only retain the Pareto optimal ones. In other words, for any F_j, if there exist another $F_{j'}$ that consumes both less time and less energy, we discard the former. Then for segment $M+1$, we do not need to worry about the solutions before segment

Algorithm 1 Offline Frequency Assignment

1: **for** each video segment i **do**
2: **for** each previous assignment $F_j = [f_{1j}, \ldots, f_{(i-1)j}]$ **do**
3: **for** each frequency f **do**
4: $F' \leftarrow$ new assignment $[f_{1j}, \ldots, f_{(i-1)j}, f]$
5: $[E', \tau'] \leftarrow$ accumulated energy consumption and time $[\sum_{n=1}^{i} E_n, \sum_{n=1}^{i} f_{ij} C_i]$
6: **if** τ' exceeds deadline **then**
7: Discard F'
8: **end if**
9: **end for**
10: **end for**
11: Sort the list of F' by E' in ascending order
12: **for** each new assignment F'_k **do**
13: **if** $\tau'_k >= \tau'_{min}$ **then**
14: Discard F'_k
15: **else**
16: $\tau'_{min} \leftarrow \tau'_k$
17: **end if**
18: **end for**
19: **end for**
20: Calculate idle energy for all F' and update E'
21: $F_{optimal} \leftarrow \{F_k | \arg\min_k E_k\}$

M. We could start searching from the Pareto optimal subset of the solution at the time, and thus reducing the amount of search needed. In the step of obtaining the Pareto optimal set of solutions, we perform an optimization by sorting all the tuples on energy, and keep track of the smallest accumulated decoding time so far as we traverse the list of tuples in ascending order of energy. In that way, if we meet any tuple that has a larger accumulated time than the smallest value in the record, we discard that tuple. At the end of the sequence of video segments, the frequency assignment that consumes the least energy will be the optimal frequency assignment.

C. Frequency Prediction

Figure 4 shows an example of optimal frequency assignment. The solid line shows the computational complexity of each segment, measured by number of CPU cycles. The shaded bars represent the actual computation with different CPU frequency, the width of which indicates the time τ_i it takes to decode ith the segment. According to our "no underflow" rule, the decoding of a segment should always finish before the time point at which it is to be played, in this case, the vertical grid lines. As marked in the figure, we use *slack*, $\Delta\tau$ to refer to the extra time left before that deadline.

Notice that at time $t = 5s$, even though the decoding workload drops by over 10%, the CPU frequency stays the same. The reason is that the slack left in that particular moment is fairly small, and the system would need a higher speed to catch up with the progress. It is precisely the points like this that provide the most precious information about how we should manage the CPU frequencies. If the next segment

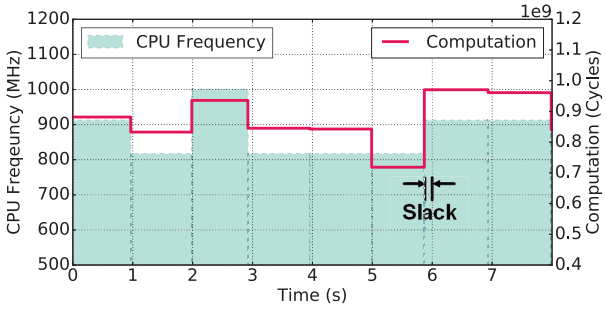

Fig. 4. An example of optimal CPU frequency assignment and the corresponding decoding process.

is predicted to have a high computational workload or the decoding buffer about to be empty, then we should switch to a higher frequency, and vice versa.

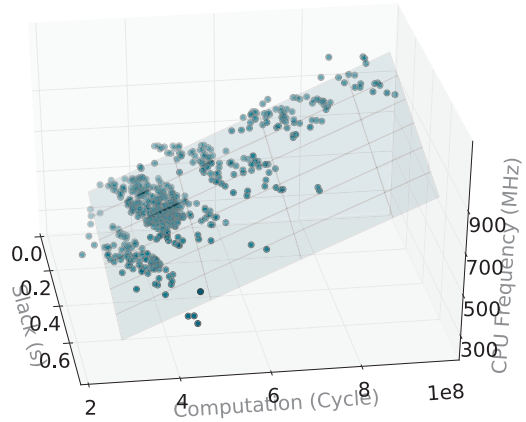

Fig. 5. Relationship between the decoding progress, the computational workload and the optimal CPU frequency assignment of the next segment. Points with darker colors are closer to the reader and those with lighter colors are further away.

To visualize the intuition, we plot the relationship between the decoding computation of the coming segment, slack and the actual frequency assigned to that coming segment in the optimal solution. The plot is shown as Figure 5. Clearly shown in the figure, the training points align well in a plane. Those points that are away from the plane are a result of the discrete frequency levels. We imagine that if there were to be more fine-grained frequency adjustment, the points would have fit even better on a plane.

Given the almost linear relationship shown in Figure 5, we decide to use linear regression as our target prediction model. With input $\mathbf{X}^T = (1, X_1, X_2, \ldots, X_p)$, linear regression predicts the output Y using the model

$$\hat{Y} = \mathbf{X}^T \beta$$

In our case, we have a simple feature vector $\mathbf{X}$ which consists of two feature: the amount of computation of the segment C_{pred}, and the slack $\Delta\tau$. Thus, our prediction model would be in the following form:

$$\hat{f_{pred}} = \beta_1 C_{pred} + \beta_2 \Delta\tau + \beta_0$$

The training data for our linear regression model consists of the statistics from the offline optimal solution. We run the solution and, for each video segment, collect computation workload, the slack it has when the decoding first began and the final choice of frequency in the optimal solution.

D. Online

In the online scenario, our assumptions for the system are the following. In the video playback system, there is a component that keeps monitoring the progress of decoding. It also keeps the records of the decoding workload of past video segments. Before we start the decoding of the next segment, we first use the method described in Section IV-A to predict the computation next segment. Then, based on the model in Section IV-C, we obtain a estimated value of the frequency $\hat{f_{pred}}$. Notice that $\hat{f_{pred}}$ will be a continuous variable, but the actual system only allow a few discrete levels of frequencies. Therefore, we choose a conservative approach where we round up the frequency to the nearest available frequency

$$f_{pred} = min\{f > \hat{f_{pred}} | f \in avaiable_freq\}$$

Once the frequency is selected, we adjust the system CPU frequency settings and wait until the decoding of the segment finishes. Then we repeat the same procedure for the following segment.

V. Evaluation

A. Experimental Setup

TABLE I
CPU Voltage and Frequency Table [3]

Volt. (mV)	750	825	900	975	1000	1050	1100
Freq. (MHz)	314	456	608	760	817	912	1000

With GEM5 [25], we are able to do cycle-accurate simulations of an ARMv7-A out-of-order processor with 32KB L1 instruction and data cache and 256KB L2 cache. Shown in table I is the voltage and frequency table of a similar processor (NVIDIA Tegra 2 [2], with ARM Cortex A9 CPU) that we obtained from the Linux kernel driver code[3]. The energy consumption is derived using McPAT [26].

We collect 50 videos that are widely used in the multimedia community to form the dataset [27]. The videos are all encoded in H.264 format with a scale of 640x480 and frame rate of 30 frames/second.

B. Optimal Frequency Assignment

Table II shows the normalized energy consumption obtained using the optimal frequency assignment, and we compare it

[2]The GEM5 simulator cannot simulate a CPU that is exactly the same as an off-the-shelf CPU such as Tegra 2, whose frequency table we are using.

TABLE II
OFFLINE NORMALIZED ENERGY CONSUMPTION

	Average	Best Case
Optimal	1.00x	1.00x
Static	1.18x	1.37x
Nominal Speed	1.67x	2.23x

against two offline baseline methods. Here "static" refers to the method where we choose the lowest voltage possible that decodes all the segments in time, and keep the same voltage throughout all the segments. The "nominal speed" refers to the method we use the nominal speed (1000 MHz) to decode and stay idle if the buffer is full. As we observed earlier, decoding workload of segments tends to stay within a small range. Thus, compared with the "static" method, we have not achieved an impressive improvement. In the cases where the decoding workload is highly variable among segments, assigning static frequency costs 37% more energy consumption. When we rush to finish decoding, the processor consumes up to 123% more energy. Since we have different assumptions about the system, we are not comparing our results with existing work in this section.

C. Diminishing Returns

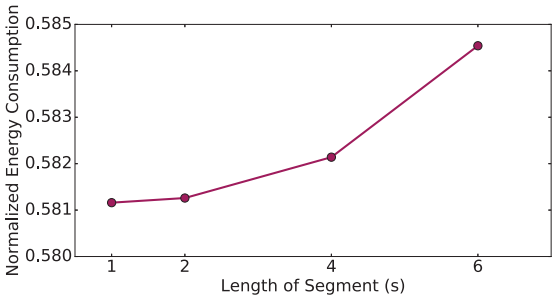

Fig. 6. The normalized energy consumption using optimal frequency assignment with different segment lengths.

Figure 6 demonstrates the point of diminishing returns. In the figure, we plot the offline optimal energy consumption when we use different segmentation length, and normalized it against the energy consumption obtained using the nominal speed (constant regardless of segment lengths). As we can see, using longer segment length, the energy consumption tends to be higher. Yet the difference in energy consumption between different segment length becomes smaller and smaller as the segments get shorter. It clearly shows the law of diminishing returns apply here: there is no need to shrink segment sizes excessively as it does not provide more benefit in terms of energy consumption.

We believe one of the contributing factors behind the findings is the limited number of supported frequency levels. Consider the extreme case where there is only one segment, and we only use one frequency throughout the whole segment. Surely there will be one optimal frequency for this scenario, but the frequency is unlikely to be among the list of supported frequencies. On the other hand, if shorter segments are used, we can approximate the optimal frequency by hopping among different frequency levels. In this sense, once we achieve the time granularity enough to approximate the frequency, there is no need for shorter segments.

D. Frequency Assignment Prediction

TABLE III
CPU FREQUENCY ASSIGNMENT PREDICTION

Mean Absolute Error	59.5 MHz
Freq. Level Prediction Accuracy	0.847

We performed a random train-test split of all the videos with a ratio of 7:3, and tested the performance of the linear regression model. Since the point of this experiment is to show the ability of to predict, the features of both the training and testing sets are derived from the statistics running the optimal frequency assignment. The prediction accuracy is shown in Table III. We measured the mean absolute error of the predicted frequency (in the continuous space) and the actual frequency assigned (in the discrete space). After we round the frequency to the nearest frequency level, we treat the results as that of a classifier and calculated the prediction accuracy. Notice that the absolute error is only around 5.95% of the highest frequency, and around 18.9% of the lowest. The level prediction accuracy is 84.7%.

E. Online Results

In the online experiment, we still test the performance using the testing video set from the train-test split, but all the predictions are performed online as the decoding is going on. There are two important aspects that we should evaluate. First, how do the energy savings in the online scenario compare with the offline optimal results. Second, will the energy savings affect user experience? In other words, we want to make sure that we don't miss the deadlines for decoding in our method.

TABLE IV
NORMALIZED ONLINE ENERGY CONSUMPTION

	Average	Best Case
Optimal	1.00x	1.00x
Our Approach	1.02x	1.00x
Nominal Speed	1.69x	2.09x

To answer the first question, Table IV shows the normalized energy consumption, compared with a few baselines. The "optimal" refers to using the offline optimal frequency and "nominal speed" refers to using the nominal speed throughout the decoding process. Our approach is able to achieve near-optimal energy savings on average, with the best case being 100% correct prediction and producing the same amount of energy savings as the optimal one. Note that since we use different datasets in training and testing, sometimes the relative performance to nominal speed is even better than the offline results.

TABLE V
MISS RATE

Buffer Underflow Rate	Buffer Overflow Rate
0.00523	0

To answer the second question, we list the buffer underflow and overflow rate in Table V. Recall that a buffer underflow will result in video segment not being ready when needed, thus harming the user experience. A buffer overflow will result in unnecessary CPU idle period, causing low energy efficiency. The experimental results show that our system only misses deadlines of 0.523% of all the video segments, and it never overflows the decoding buffer.

To more intuitively understand how the proposed system performs in the online scenario, we show a few case studies in the next subsection.

F. Case Studies

Figure 7 shows the 4 typical cases of online frequency assignment. Similar to the previous figures, the dotted line represents the decoding workload, the bar with the dashed edge is the optimal frequency assignment and the solid line represents the predicted frequency assignment. Since the predictions happen online and are sensitive to the previous state, it is high unlikely that the predicted frequencies are the same as the original optimal frequencies. However, our predictions show overall similar patterns to the original ones: if the optimal solution uses a higher frequency at t, then the predicted solution is very likely to switch to higher frequencies at the vicinity of t (e.g. $t-1$ or $t+1$). Aside from the similarity, these cases each represents a specific scenario. Now we will discuss them separately.

Figure 7(a) shows the online decoding process of the video `carphone`, and is one of the cases where the prediction is very close to the actual optimal solution. The video features a man talking while driving in a car. Notice that the 6th segment has noticeably higher requirement for computation. This is due to a background scenario change outside of the car. However since we already built up some decoding buffer earlier (notice at the end of the 3rd second, we are further ahead than the optimal solution), our prediction module chose to stay with the same frequency.

Figure 7(b) shows the online solution for video `waterfall`. An interesting fact that we noticed here is that the frequency of the last segment is usually mispredicted. Our predictor is not smart enough to know that it is the last segment and it has the freedom to use up all the slack time available. Thus, the predicted solution usually ends earlier, resulting a small portion of energy waste. In this case, the online solution consumes about 1.040 times the energy of the optimal solution (the average case being 1.02x).

The video `tempete` has rather high amount of variations in the decoding workload, as shown in Figure 7(c). The hard part comes when we have a sudden rise in the amount of computation, which is what happens when we try to decode segment number 6 and 8. In the optimal solution, we see two distinct peaks at the corresponding video segments. In our online solution, what the system reacted in response to the peaks are the two very high CPU frequencies used right after the peak. Recall that our frequency prediction is based on the estimated computation. The estimated computation is high after the given peak in computation. In addition, we have a relatively small buffer by the end of the peak, especially in the second case. Because of those reasons, the system decides to use very high voltage for the decoding of the next segment. Notice that we nearly missed the deadline for the second peak. But fortunately, as we have over half second slack time in our buffering, we were able to avoid missing the deadline.

The last case, showed in Figure 7(d), involves the decoding of a video where there is a sudden drop in decoding workload. It actually corresponds to the video `pamphlet` shown in Figure 3. Fortunately, a drop in computation is much easier to handle than the previous case. The response of the system is that for segment number 6, a mistakenly high frequency is predicted, resulting in a huge slack time. However, we ended up not suffering too much from that single misprediction. The overall online system energy consumption is 1.026 times of the optimal solution.

VI. CONCLUSION

In this paper, we discussed an application level DVFS method for video decoding. We prose a machine learning based prediction model for CPU frequency selection using the estimated workload and decoding progress as its features. We show that with application semantics, the system is able to perform DVFS at a low rate that satisfies constraints imposed by existing operating systems, while saving an average of 40.1% energy consumption compared with execution at nominal speed.

VII. ACKNOWLEDGEMENT

This work was supported in part by the NSF under Award CNS-0958369 and Award CNS-1059435.

REFERENCES

[1] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pp. 197–202, 1998.
[2] M. Pedram and J. M. Rabaey, *Power aware design methodologies*. Springer Science & Business Media, 2002.
[3] "Linux kernel." https://www.kernel.org/doc/. Accessed: 2016-02-27.
[4] "Android operating system." https://android.googlesource.com/. Accessed: 2016-05-15.
[5] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps," in *The 10th International Conference on Mobile Systems, Applications, and Services, MobiSys'12*, pp. 267–280, 2012.
[6] P. Tseng, P. Hsiu, C. Pan, and T. Kuo, "User-centric energy-efficient scheduling on multi-core mobile devices," in *The 51st Annual Design Automation Conference, DAC*, pp. 85:1–85:6, 2014.
[7] J. e. a. Hopper, "Using the linux cpufreq subsystem for energy management," *IBM blueprints*, 2009.
[8] P. Mochel, "The sysfs filesystem," in *Linux Symposium*, p. 313, 2005.

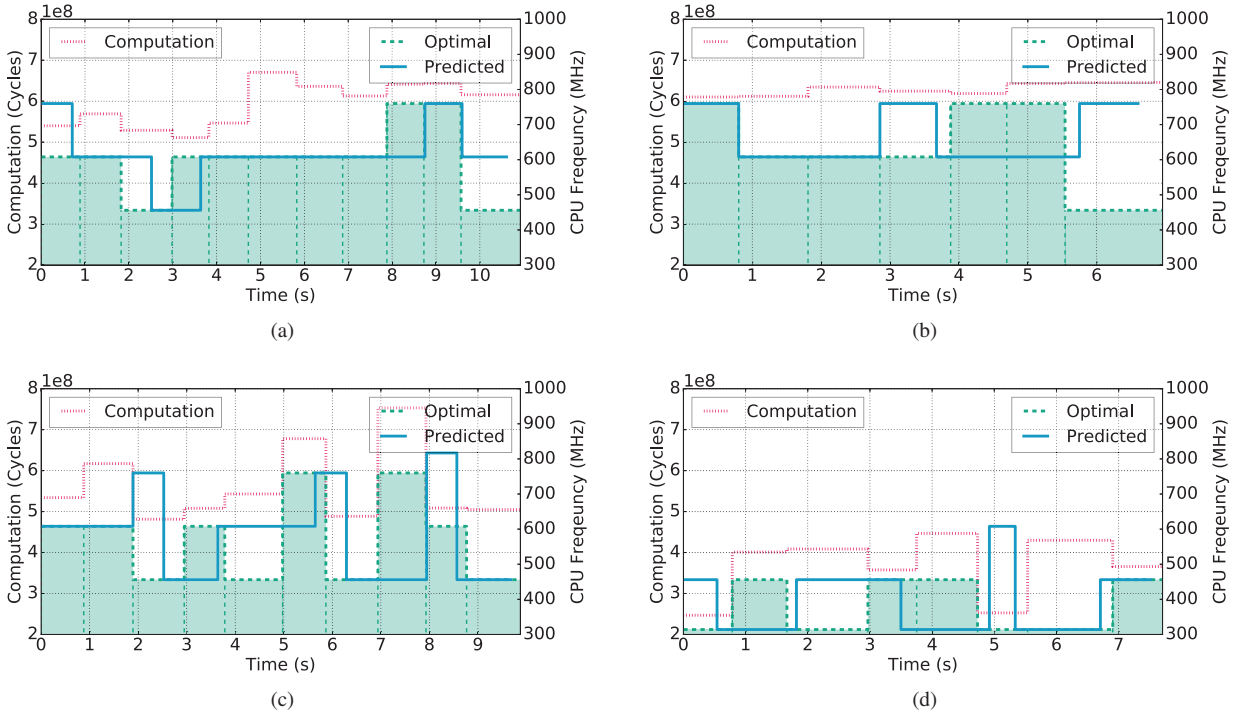

Fig. 7. Figures showing the offline (optimal) and the online (predicted) CPU frequency assignment to segments in the test video 7(a) `carphone`; 7(b) `waterfall`; 7(c) `tempete`; 7(d) `pamphlet`.

[9] D. Hillenbrand, Y. Furuyama, A. Hayashi, H. Mikami, K. Kimura, and H. Kasahara, "Reconciling application power control and operating systems for optimal power and performance," in *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–8, 2013.

[10] 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, ICCAD*, pp. 732–737, 2002.

[11] Z. Ma, H. Hu, and Y. Wang, "On complexity modeling of H.264/AVC video decoding and its application for energy efficient decoding," *IEEE Trans. Multimedia*, vol. 13, no. 6, pp. 1240–1255, 2011.

[12] J. Hamers and L. Eeckhout, "Exploiting media stream similarity for energy-efficient decoding and resource prediction," *ACM Trans. Embedded Comput. Syst.*, vol. 11, no. 1, p. 2, 2012.

[13] A. Carroll and G. Heiser, "Unifying DVFS and offlining in mobile multicores," in *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pp. 287–296, 2014.

[14] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 286–292, 2015.

[15] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *The 51st Annual Design Automation Conference, DAC*, pp. 170:1–170:6, 2014.

[16] J. Chen and C. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 28–38, 2007.

[17] R. Xu, D. Mossé, and R. G. Melhem, "Minimizing expected energy consumption in real-time systems through dynamic voltage scaling," *ACM Trans. Comput. Syst.*, vol. 25, no. 4, 2007.

[18] J. R. Lorch and A. J. Smith, "PACE: A new approach to dynamic voltage scaling," *IEEE Trans. Comput.*, vol. 53, no. 7, pp. 856–869, 2004.

[19] 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*, pp. 149–163, 2003.

[20] E. Chung, G. D. Micheli, and L. Benini, "Contents provider-assisted dynamic voltage scaling for low energy multimedia applications," in *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pp. 42–47, 2002.

[21] P. Kumar and M. B. Srivastava, "Power-aware multimedia systems using run-time prediction," in *14th International Conference on VLSI Design (VLSI Design 2001), 3-7 January 2001, Bangalore, India*, pp. 64–69, 2001.

[22] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pp. 34–39, 2001.

[23] Y. Lu, L. Benini, and G. D. Micheli, "Dynamic frequency scaling with buffer insertion for mixed workloads," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 11, pp. 1284–1305, 2002.

[24] M. Kim, Y. G. Kim, S. W. Chung, and C. H. Kim, "Measuring variance between smartphone energy consumption and battery life," *IEEE Computer*, vol. 47, no. 7, pp. 59–65, 2014.

[25] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[26] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, pp. 469–480, 2009.

[27] "xiph.org." http://www.xiph.org/. Accessed: 2016-06-30.