# Dynamic Power Optimization of Interactive Systems

Lin Zhong and Niraj K. Jha
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544
{lzhong, jha}@ee.princeton.edu

*Abstract*— **Power has become a major concern for mobile computing systems such as laptops and handhelds, on which a significant fraction of software usage is interactive instead of computation-intensive. An analysis shows that over 90% of system energy and time is spent waiting for user input. Such idle periods provide vast opportunities for dynamic power management (DPM) and voltage scaling (DVS) techniques to reduce system energy. The user interface is in charge of system-user interaction. It often has *a priori* knowledge about how the user and system interact at a given moment. In this work, we propose to utilize such *a priori* knowledge and theories from the field of Psychology to predict user delays. We show that such delay predictions can be combined with DPM/DVS for aggressive power optimization. We verify the effectiveness of our methodologies using usage traces collected on a personal digital assistant (PDA) and a system power model based on accurate measurements. Experiments show that using predicted user delays for DPM/DVS achieves an average of 21.9% system energy reduction with little sacrifice in user productivity or satisfaction.**

## I. Introduction

Power has become a dominant concern for mobile computing systems. While previous power management techniques were mostly concerned with computation-intensive and reactive applications, we are more interested in interactive applications that are ubiquitous on modern mobile computing systems.

In this work, we analyze the power characteristics of interactive systems, and employ the user interface information, history, and Psychology theories to predict user delays during system-user interaction. We show that such delay predictions can be utilized by DPM/DVS techniques to reduce system energy consumption very effectively. As far as we know, this is the first work that addresses power optimization of interactive systems from the perspective of system-user interaction. Unlike other works, in which energy reduction is reported only for the processor, we report energy reduction for the whole system.

The paper is organized as follows. After discussing background and related work in Section II, we detail the model we employ for system-user interaction in Section III. User-delay models based on Psychological theories and history are proposed in Sections IV and V, respectively. We show how DPM/DVS techniques can take advantage of predicted user delays to save energy in Section VI. We then present our benchmarks and experimental results in Section VII. Next, we present discussions in Section VIII and conclusions in Section IX.

## II. Background and Related Work

We first provide energy usage characteristics of an example interactive software, and then discuss background and related work on power optimization of interactive systems, DPM/DVS, and system-user interaction modeling.

### A. Energy characteristics of interactive systems

Fig. 1 shows the power consumption when the Qtopia [23] Calculator is used by a user on a Linux-based Sharp Zaurus SL-5500 PDA, to compute $(89 \times 56) \div 45$. The power is sampled at a rate of 400 samples per second.

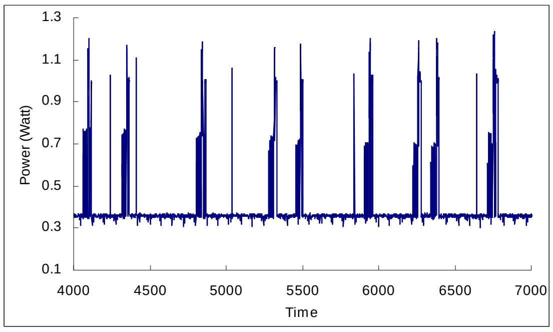

Fig. 1. Power consumption of Sharp Zaurus while running Calculator.

In Fig. 1, there are power valleys separated by nine major power peaks, which correspond to software responses to user tapping of GUI buttons. In the power valleys, the system waits for user input while the Linux kernel does maintenance jobs like handling timer interrupts and scheduling, which introduces small fluctuations and several minor spikes in the valley. Such power characteristics are typical of most interactive software usage.

To see how much time and energy the valleys take, we analyzed usage traces for two users of four commercial applications shipped with the PDA, as detailed in Section VII. The percentage of total time and energy the PDA spent waiting for user input is shown in Table 1. Clearly, over 90% of the time and energy was spent in waiting for user input. Moreover, most of the waiting periods are longer than 500ms. This demonstrates the vast opportunities available for power optimization of such interactive systems.

### B. Power optimization for interactive systems

Most previous system power optimization work has been CPU-centric and devoted to reducing the system busy energy, which can be seen to be minuscule for the applications shown in Table 1. Since the user interface, and hence the display, must be active while the system waits for user input, reducing user interface and display-related energy can be quite effective for interactive systems [5, 8, 28]. Since interactive systems spend so much time and energy waiting, it is extremely important to accelerate software usage through user interface designs to achieve energy efficiency [28].

Acknowledgments: This work was supported by DARPA under contract no. DAAB07-02-C-P302.

Table 1
Percentage of system time and energy spent waiting for user input

| Benchmarks | User | Total time (s) | Time(%) | Energy(%) |
|---|---|---|---|---|
| Calculator | 1 | 39.2 | 99.4 | 98.5 |
| | 2 | 17.4 | 99.8 | 97.8 |
| Filebrowser | 1 | 187.5 | 99.1 | 97.6 |
| | 2 | 106.3 | 98.7 | 96.4 |
| Go | 1 | 1,214.9 | 97.9 | 94.2 |
| | 2 | 258.9 | 94.6 | 90.2 |
| Solitaire | 1 | 734.3 | 99.8 | 99.6 |
| | 2 | 397.2 | 99.1 | 97.4 |

## *C. Dynamic power management and voltage scaling*

DPM/DVS techniques are concerned with changing system performance levels dynamically to reduce system energy consumption, and have been extensively investigated [2,15]. One of the key problems in DPM/DVS is resource usage prediction. Various predictive and stochastic techniques have been proposed [2]. However, as demonstrated in [9, 21], most existing resource usage prediction methods may work well for computation-intensive tasks, but not for interactive tasks. Although many works [7, 9, 16, 21] have used interactive applications as their benchmarks, they have treated these applications in the same fashion as computation-intensive applications and have not exploited any features of interactive applications to make resource usage predictions better. Some of them [7,16] have proposed to utilize the human-perceptual limit to slow down the system so that the user interface can respond within the limit. Since many user interface operations still require relatively high performance, the proposed technique missed the opportunity to further slow down the processor during its wait time.

In [25], an exponential cumulative distribution was used to model user requests for power management. This is stochastic and too high-level to accurately model actual user delays.

## *D. System-user Interaction Modeling*

Computer-human interaction has been jointly investigated by Computer Scientists and Psychologists. Research in this field provides insights into user response time. A recent survey of user response time models can be found in [17]. Specifically, system-user interaction with menus has been studied in [12, 20], and usage of a stylus and soft keyboard has been studied in [24]. Research on human reading speed is discussed in [4]. Another important aspect of system-user interaction modeling is user interface specification. Many formal models have been used for this purpose, such as state-transition diagrams (STD) [27], Petri nets [19], goal-operation-methods-selections [3], and user action notation [10].

## III. System-user Interaction Modeling

A user interface defines the framework for system-user interaction. That is, both the user and software only have predefined and limited choices for how they can proceed. More importantly, a user interface knows exactly when a user delay will occur and provides critical information for how long it will last. In this section, we utilize the STD model [27] to use such knowledge. In the next two sections, we propose models to predict the length of user delays based on this knowledge.

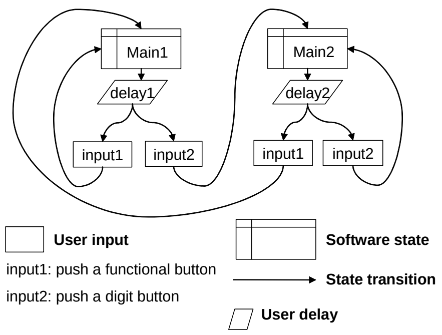

Fig. 2. An STD for Calculator.

## *A. STD extraction*

A user interface is sometimes designed and specified as an STD, in which case there is no need for STD extraction. Otherwise, an STD can be usually extracted from the interface implementation with little difficulty. The STD we extracted for the Qtopia Calculator is shown in Fig. 2.

Most user interfaces are implemented using objects and event handlers associated with objects. At a given time, a subset of the objects is able to accept certain events. Since an event handler processes user inputs and generates new information for the user, it forces the user interface into a state, which can be the same as or different from the original one. Therefore, event handlers with the corresponding triggering user inputs and events correspond to the transition arcs of the STD. A user delay is incurred as soon as the event is generated. Therefore, the user interface knows when user delays occur. If we assume an event handler transitions the user interface into a unique state, an STD is naturally defined by the event handlers. Such an STD is actually the most detailed STD that can be extracted for the object/event handler based implementation. It provides the complete system-user interaction description. Note that an event handler in the implementation may correspond to more than one transition arc since the same event handler can be activated from different states. Therefore, all the arcs entering a state correspond to the same event handler.

If we only care about some specific user delays during system-user interaction or it is hard to infer the complete STD, we can identify event handlers which lead to important states only. Instead of a complete STD, we will then have a set of its snapshots. Since the user delay of a state is always associated with the corresponding event handler, such a set of snapshots is still useful for knowing when and how long these user delays will be. Moreover, the complete STD can be compacted by clustering similar states into a single state, based on content or profiling. For content-based clustering, let $\Psi(s)$ denote the set of objects for state $s$ which are able to accept events, and $\Omega(o, s)$ denote the set of events that object $o \in \Psi(s)$ is able to accept. We define the *content set* for state $s$ as

$$\Xi(s) = \{(o, \Omega(o, s)) | o \in \Psi(s)\}$$

Two states can be combined into one if their content sets are the same. For example, the two states in Fig. 2 can be clustered into a single state.

## *B. Intra-state consistency*

For a user, an STD state represents a unique and relative consistent user interface and requires relatively consistent user reaction. However, different states differ in their intra-state consistency. The same state may have different internal data and external user interface appearance during usage. The content of a state can even be dynamic, *e.g.*, file browsers and text readers. Intra-state consistency can be improved by using more states. The dynamic content problem can be partially solved by obtaining information about the content, such as obtaining the number of items in the current directory and the size of the text to be shown.

## IV. User Delay Models from Psychological Studies

If the content of the user interface of an STD state is known, the length of the user delay can be predicted based on it. There are three basic processes involved in a user's response to a system according to the *model human processor* [3]. They are the perceptual, cognitive, and motor processes. Before the user physically touches the system input peripherals, the system is idle, except that it may display a flashing cursor to indicate the current location for user input, which can be disabled in most applications without much sacrifice in user-friendliness.

### A. Perceptual delay

Psychological studies [4] have shown that humans read through discrete eye movements, which consist of fixations and saccades. Fixations are the state in which the eyes are focused on a object statically. Saccades represent rapid eye movements from one fixation location to another. Information is absorbed only during fixation, which lasts from 60 to $700ms$. A saccade takes about $30ms$.

We categorize modern computer display information into two categories: graphical and textual. Graphical information is presented through graphical objects which consist of 1) window widgets, such as buttons, radiobuttons, menus, *etc.*, and 2) pictures or figures. To simplify delay estimation, we assume each graphical object requires one separate fixation and each fixation lasts for at least $60ms$. Therefore, for each graphical object, a delay of $90ms$ (one minimal fixation plus one saccade) is added.

Text can be regarded as a special kind of graphical object. Psychological studies [4] have shown that college students can raud (read with comprehension) at a typical rate of 300 standard-length words per minute or five per second. The standard-length word was assumed to have six characters. For a text with $n$ words, the typical time for rauding is

$$T = \frac{n\times cpw}{Wpm \times 6} = \frac{n\times cpw}{30}(s)$$

where $cpw$ is the average number of characters per word and $Wpm$ is the rauding rate in terms of the number of standard-length words per minute. If a text is associated with a graphical object, the reading time is used as the fixation length for the graphical object and there is no separate delay for text reading.

### B. Cognitive delay

The Hick-Hyman Law [11, 13] states that the time to make a decision based on $N$ distinct choices is given by

$$RT = a + b\log_2 N(s)$$

Parameters $a$ and $b$ are constants that can be empirically determined. Based on the information in [3], we assume $a$ to be 0 and $b$ to be $\frac{1}{7}$.

The Hick-Hyman Law can be useful only if the number of different choices, $N$, is known. Unfortunately, it is hard or even impossible to estimate $N$ for many cognitive processes. A menu selection may actually involve more choices for a user than the number of menu items because the user may evaluate the consequence of selecting an item and the possible subsequent choices to decide whether to select that item or not. In this study, we conservatively assume $N$ to be the number of most direct choices like the number of menu items and the number of buttons to choose from.

### C. Motor delay

It also takes time for a user to make a physical movement. In most handheld computers, users respond by touching some target: either physical key(s) or user interface objects on the touch-screen. We only consider the usage of user interface objects. The Fitts Law [6] states that for a normal human being, the motor time to carry out a movement is related to the size of the target and the distance to the target as

$$MT = 0.23 + 0.166 \cdot \log_2(\frac{A}{W} + 1)(s)$$

where $A$ is the amplitude of the movement (assumed to be one quarter of the screen height in this work) and $W$ the width of the target as measured in the direction of motion [18]. We have adopted the values of coefficients from [18].

The above three models are together referred to as the *psychological model* in this work. Perceptual and motor delays can be predicted relatively well. However, the cognitive delay can be only pessimistically predicted based on the Hick-Hyman Law, as mentioned before. Therefore, the psychological model is used to predict the lower bound on user delay. In the next section, we propose a history-based user delay model to predict the actual delay.

## V. History-based User Delay Model

The psychological model is content-based. It predicts user delay length knowing what users will see. In some cases, it may be difficult to know beforehand the user interface content. Therefore, we devise a user delay model based on history, *i.e.*, multiple observations from the past. It works in a fashion similar to the median filter in order to filter out randomness in user behavior. Let $D^i$ denote the $i$th last observed delay for an STD state and $M$ the total number of user delays recorded for that state.

The delay record is sorted to generate a new series so that

$$D_s^1 < D_s^2 < ... < D_s^M$$

Then we define the *habitual set* $\Phi(p)$ as

$$\Phi(p) = \{D_s^{\lfloor M\cdot p\rfloor+1}, D_s^{\lfloor M\cdot p\rfloor+2}, ..., D_s^{\lfloor M\cdot(1-p)\rfloor}\}$$

where $0{\le}p{\le}0.5$ with a typical value of 0.25. $\Phi(p)$ contains the central $M{\cdot}(1-2{\cdot}p)$ items of the sorted series. It is a set extension of the concept of a median. Let $m_\Phi$ denote the mean of $\Phi(p)$. Then the next user delay for the state is predicted as

$$D = \alpha{\cdot}m_\Phi$$

where $0 < \alpha < 1$. $\alpha$ is called the *pessimism* factor.

When $p = 0.5$, the above approach reduces to using the median of the delays in the recorded history. When $p = 0$, it reduces to using the mean instead.

## VI. Utilizing User Delays for DPM/DVS

In the previous sections, we showed how to determine when a user delay occurs and predict its length. Such resource usage predictions are exactly what are needed for DPM/DVS to be effective. Although DPM/DVS is an orthogonal problem to user delay modeling, we next show how user delays can be used to reduce system energy consumption through DPM/DVS.

## A. Application-directed DPM/DVS

For simplicity, we assume there is only one interactive application under consideration, which is usually true for handheld computers. Also, we assume application programming interfaces (APIs) are available to applications to change the system performance levels. The flowchart for inserting the performance level transition code is given in Fig. 3(a). First, the STD of the user interface is extracted, and major user interface event handlers/states are identified, through either post-implementation analysis or pre-implementation specifications. If the psychological model is used, the user interface content in each state is parsed to get the values of the parameters for user delay models, as described in the previous sections. The parameter values only depend on the current state. Based on the predicted delay, an appropriate performance level is selected for the state and the corresponding code is inserted at the end of the corresponding event handler.

The inserted code can either force the system back to a higher performance level after a time interval based on the predicted user delay, or just let the user input raise the performance level, depending on the delay overhead for performance level transition and user's delay tolerance.

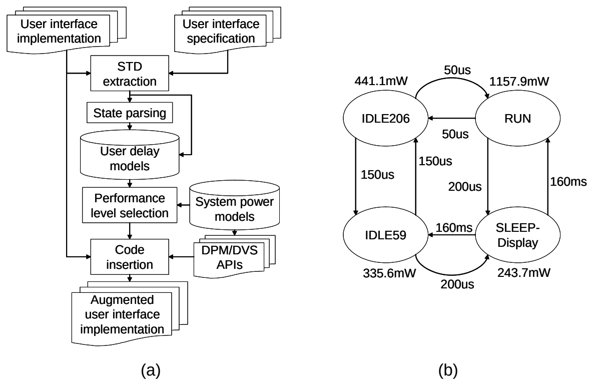

Fig. 3. (a) The flowchart for code insertion for DPM/DVS, and (b) power modes and mode transitions.

## B. Theoretical energy savings

We next present a theoretical analysis of the energy savings obtained when DPM/DVS is combined with user delay modeling.

For simplicity of exposition, consider two performance levels of the system from a pool of performance levels for a given user delay. Let $D$ denote the user delay, $P$ the system power consumption for the higher performance level, and $P'$ that for the lower one. The energy consumption for the system during the user delay without DPM/DVS will be

$$E = D \cdot P$$

Suppose we can predict the user delay perfectly, change the system performance to the lower one after the system finishes responding, and change it back to the higher one right before the next user input. Let $P_{HL}$ and $P_{LH}$ denote the performance level transition power, and $T_{HL}$ and $T_{LH}$ denote the delay for higher-to-lower and lower-to-higher transitions, respectively. We also assume that $D > (T_{HL} + T_{LH})$. Then the energy consumption for the system during the user delay with such a performance-level transition is given by

$$E_0 = (D - T_{HL} - T_{LH}) \cdot P' + T_{HL} \cdot P_{HL} + T_{LH} \cdot P_{LH}$$

The energy saving is therefore

$$\Delta E_0 = E - E_0 = D \cdot (P - P') - T_{HL}(P_{HL} - P') - T_{LH}(P_{LH} - P')$$

If we assume $P = P_{HL} = P_{LH}$, we have

$$\Delta E_0 = \{D - (T_{HL} + T_{LH})\} \cdot (P - P')$$

The energy saving ratio is calculated as

$$\rho_0 = \frac{\Delta E_0}{E} = \{1 - \frac{(T_{HL} + T_{LH})}{D}\} \cdot (\frac{P - P'}{P})$$

If the system changes the performance level upon a user input, we can obtain the energy saving through a similar analysis as

$$\rho' = \rho_0 - \frac{T_{LH}}{D} \cdot \frac{P'}{P}$$

If the delay models are used to predict $D$ as $D'$ and the system is put into the low performance level if $D' > (T_{HL} + T_{LH})$ and put back into the high performance level right before the predicted user delay elapses, we have

$$\rho = \begin{cases} \rho_0 - \frac{T_{LH}}{D} \cdot \frac{P'}{P}, & D' \geq D + T_{LH}; \\ \rho_0 - \frac{|D - D'|}{D} \cdot \frac{P'}{P}, & D \leq D' < D + T_{LH}; \\ \rho_0 - \frac{|D - D'|}{D} \cdot (1 - \frac{P'}{P}), & (T_{HL} + T_{LH}) \leq D' < D; \\ 0, & D' \leq (T_{HL} + T_{LH}). \end{cases}$$

Given the two performance levels, the energy saving ratio is only dependent on the user delay and prediction error. Comparing $\rho$ with $\rho'$, we can see that using predicted user delay is actually more energy-efficient if $T_{LH}$ is large compared with user delay prediction errors. Note that both $\rho$ and $\rho'$ can be negative, which means energy consumption can be actually increased if performance-level transition is not properly done.

## VII. Benchmarks and Experimental Results

In this section, we will use system-user interaction traces collected from real software usage to show the benefits of DPM/DVS for user delays. Our techniques are applicable under any power management method that transitions from one performance level to another. Both DPM and DVS belong to this category.

### A. Benchmarks

We considered four applications from Qtopia version 1.5.0 on a Linux-based Sharp Zaurus SL-5500 PDA as our benchmarks. They are Calculator, Filebrowser, Go, and Solitaire. Calculator is as described in Section II. Filebrowser is a GUI application for listing files in the local storage. Go and Solitaire are two popular games well known by their names.

Based on an analysis of their source code, STDs were extracted for these applications. The STDs for Calculator (as shown in Fig. 2) and Filebrowser have two and five states, respectively. The STDs for Go and Solitaire only have one state, in which the game waits for the next user move.

### B. Usage trace collection

System-user interaction traces were collected on a Sharp Zaurus SL-5500 PDA. We inserted *gettimeofday* at the beginning and end of the event handlers in the benchmark applications and recorded the timing. This record traces when the system finishes its response and when it starts processing an incoming event and changes state, triggered by a user input. The state name, actually, the event handler name, is also recorded. With regard to accuracy, *gettimeofday* reports time in microseconds. Moreover, according to the timing method, the delay of the system in generating an event after receiving a user input is counted as part of the user delay instead of the system busy time. In our experience, the error this introduces is negligible compared to the delay for event processing.

Two users participated in trace collection. One is a male graduate student majoring in Engineering while the other is a female graduate student majoring in Social Sciences. Both are veteran computer users. Before trace collection, they were already familiar with real calculators and the usage of software similar to the file browser. The male user was also familiar with the Patience game in Solitaire. The female user was not. Neither of them knew how to play Go. They were given oral instructions on how to operate the PDA and use the benchmark applications. They were also given time to play with the benchmark applications on the PDA to get acquainted with them. Then they were asked to complete the tasks as described in Table 2 in an office environment.

Table 2
Tasks for usage trace collection

| Benchmark | Task |
|---|---|
| Calculator | Compute a formula with three two-digit numbers and two operations |
| Filebrowser | Find a given file in the local storage |
| Go | Play a new game for several minutes |
| Solitaire | Play a new Patience game for several minutes |

Each user was asked to perform the tasks once per day and for a total of three days. Therefore, for each user and task, three traces were collected. The total length of time of traces for each benchmark and user is given in Table 1.

The benchmark source code for trace collection can be downloaded from [1].

## C. System power model

The Sharp Zaurus SL-5500 PDA has an Intel StrongARM SA-1110 processor [14], which can run in three modes: RUN, IDLE, and SLEEP. Moreover, its core clock speed can be varied between 59 and $206MHz$ in discrete steps. It is put into the IDLE mode by the Linux kernel whenever there is nothing to run. We constructed a system power model based on power measurements and available industrial data. The measurement equipment is the same as that used in [28]. The modes of system operation and their corresponding power consumptions are shown in Fig. 3(b). Permitted mode transitions are shown as directed arcs. The delay overhead for mode transitions is as marked on the directed arcs between modes [14]. The front-light is assumed to be always off and the display is assumed to be always on with constant power consumption of $234.4mW$. We assume the power consumption is constant for an operation mode. Notably, the display consumes about half of the system idle power.

RUN corresponds to the SA-1110 RUN mode when the processor is busy executing instructions. Its power is measured when the PDA computes discrete cosine transforms, as detailed in [28]. IDLE206 and IDLE59 correspond to the SA-1110 IDLE mode with a core clock frequency/voltage of $206MHz$/1.5V and $59MHz$/1.25V, respectively. IDLE59 is a hypothetical mode for the Zaurus SL-5500, but is real for many other SA-1100 or SA-1110 based systems [9, 22, 26]. It is estimated as the display power plus the measured non-display power at $206MHz$ scaled down using the same ratio of the power for IDLE59 to that for IDLE206 in the Itsy system power measurement presented in [26]. SLEEP-Display corresponds to the SA-1110 SLEEP mode when the display is left on. Therefore, its system power consumption is measured when the system is suspended, and the display power is added thereafter.

The energy consumption for system usage is obtained by running its trace through the system power model with DPM/DVS. There are different ways to do DPM/DVS with such a system power model in view of user delays, as discussed next.

The Linux kernel automatically puts the system into the IDLE206 mode whenever there is no process running and returns it to the RUN mode upon interrupts. We take this as the baseline and report the performance of other techniques against it.

The most straightforward DPM/DVS technique would be to put the system into the IDLE59 or the SLEEP-Display mode right after the system finishes responding to the user and put it back into the RUN mode upon a user input. These methods are called the *simple* and *lazy* techniques, respectively. Since the IDLE59 to RUN mode-transition delay is small, there is no concern with regard to user productivity. However, that for SLEEP-Display to RUN will most likely be noticed by users and decrease user productivity.

Assuming we can predict user delays absolutely accurately, we can choose to put the system into the SLEEP-Display mode and wake it up right before the next user input, if the delay is long enough. This is called the *perfect* technique since it gives the upper bound on energy savings based on the system power model.

The proposed user delay models can be used to predict user delays. There are two concerns with respect to prediction errors. First, in the case of overestimation, the system will wake up from the SLEEP-Display mode upon an interrupt generated by the user input and enter the RUN mode directly. Such errors are called *lazy errors*. If the delay is overestimated by more than the human perceptual threshold, the user will experience a noticeable mode transition delay. Such lazy errors are called *serious lazy errors*. Second, in the case of underestimation, the system wakes up and transfers to the IDLE59 mode after the predicted delay to get ready for the user input and will not be able to fully exploit the idle time to reduce energy by remaining in the SLEEP-Display mode. Therefore, we report the average error for underestimations. We refer to the DPM/DVS technique based on user delay predictions by the history-based and psychological models as *history* and *psychological*, respectively.

## D. Energy savings

In Table 3, we give energy savings (ES) against the baseline. These include the simple, lazy, perfect, psychological, and history techniques. In the history technique, the pessimism factor $\alpha$ is assumed to be 0.4 and the last seven delay records are used starting with the psychological predictions. The average underestimation error (AUE) (the fraction by which the user delay is underestimated), percentage of lazy error (PLE) (the percentage of all predictions that are over-estimations) and percentage of serious lazy error (PSLE) are also reported for the psychological and history techniques. We adopt $50ms$ as the human perceptual threshold.

Table 3

Energy savings based on predicted user delays

| Benchmark | User | Simple | Lazy | Perfect | Psychological | | | | History(0.4) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ES(%) | ES(%) | ES(%) | ES(%) | PLE(%) | PSLE(%) | AUE(%) | ES(%) | PLE(%) | PSLE(%) | AUE(%) |
| Calculator | 1 | 17.9 | 18.8 | 35.1 | 27.3 | 0.0 | 0.0 | 28.4 | 20.1 | 0.0 | 0.0 | 64.5 |
| | 2 | 17.9 | 15.3 | 34.3 | 22.9 | 0.0 | 0.0 | 39.9 | 19.3 | 0.0 | 0.0 | 60.5 |
| Filebrowser | 1 | 17.6 | 32.9 | 37.7 | 23.7 | 0.0 | 0.0 | 51.3 | 21.8 | 2.7 | 2.7 | 64.8 |
| | 2 | 17.7 | 34.4 | 38.2 | 21.0 | 0.0 | 0.0 | 61.9 | 19.4 | 0.0 | 0.0 | 73.9 |
| Go | 1 | 17.5 | 36.0 | 38.2 | 18.9 | 0.5 | 0.0 | 77.5 | 21.4 | 10.6 | 9.7 | 66.5 |
| | 2 | 15.8 | 25.3 | 32.9 | 18.9 | 2.3 | 1.2 | 59.9 | 19.0 | 1.7 | 0.0 | 63.2 |
| Solitaire | 1 | 18.2 | 35.2 | 39.3 | 22.8 | 3.7 | 2.9 | 61.5 | 22.8 | 7.9 | 6.2 | 64.8 |
| | 2 | 17.9 | 32.4 | 38.0 | 19.6 | 8.7 | 5.5 | 68.9 | 21.4 | 14.2 | 12.0 | 65.7 |

Notably, Lazy's energy savings are closest to Perfect's in three of the four benchmarks. Only in Calculator, it is less energy-efficient than the other techniques due to the fact that user delays for Calculator tend to be much shorter and predictable (see Section VI). However, its PSLE is almost 100%. Therefore, Lazy will be useful only when system delay is tolerable and user delay is relative long, as in the case of map viewer and text reader applications.

For the psychological technique, the PLE is very low since this model is pessimistic. However, for the history technique, the pessimism factor controls the tradeoff between the PLE and ES. Fig. 4 shows how the ES changes with PLE for Calculator and Filebrowser as the pessimism factor varies from 0.2 to 1.3 for the history technique (His). It also shows the tradeoff point for the psychological technique (Psy). To achieve the same energy saving as the psychological technique, the history technique needs to make a lot more lazy errors. This demonstrates the superiority of the psychological technique when avoiding lazy errors is important.

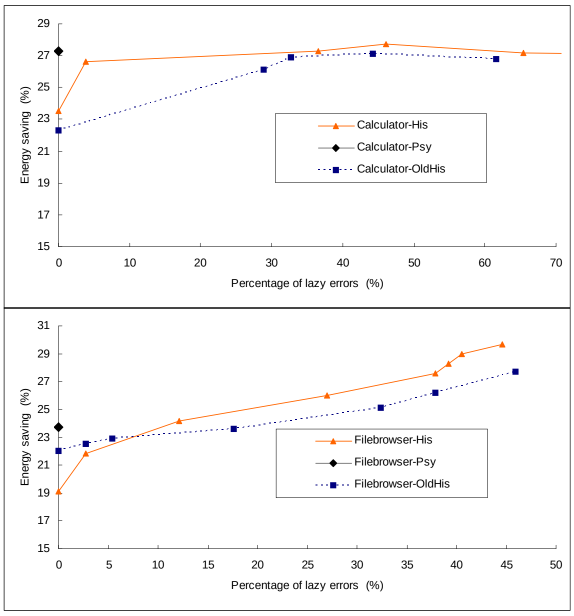

Fig. 4. Tradeoff between percentage of lazy errors and energy savings for Calculator and Filebrowser.

To show the benefit of using user interface information in the history technique, Fig. 4 also shows the tradeoff curve (OldHis) for the conventional history-based idle time prediction technique, which uses the last seven observed delays to predict the next delay without regard to STD states. It is clear that STD state-aware delay prediction is better than conventional prediction. Moreover, the Calculator and Filebrowser's user interfaces are relatively simple. We expect the advantage of using state-aware delay prediction to be more for more complicated user interfaces.

It is worth noting that the psychological technique performs better for Calculator and Filebrowser than Go and Solitaire compared to the history technique in terms of ES, AUE, and PLE/PSLE. This is due to the fact that operating Calculator and Filebrowser is cognitively much simpler and their cognitive processes are better modeled by the Hick-Hyman Law.

## VIII. Discussions and Future Work

In the previous section, we showed the effectiveness of combining DPM/DVS with user delay prediction. There are several points worth discussing.

### A. *STD extraction*

As may be apparent to the reader, how an STD is extracted for an interactive software is important for user delay prediction. For example, if Calculator is modeled using only one state, which may seem like a natural model, the user delay prediction may degrade as demonstrated for the history-based model. Also, the not-so-good performance of user delay models for Go and Solitaire can be partially attributed to modeling them with a single-state STD. Adding states into the STD has the potential to improve intra-state consistency and the tradeoff between energy savings and lazy errors.

### B. *Other ways to benefit from user delay prediction*

We showed how user delay prediction can be utilized for DPM/DVS in the scenario in which there is only one application, the interactive one, under consideration. Moreover, the system performance level was set by this application. In some other scenarios, an interactive application may be used with other applications running in the background. A user may enjoy music using the software MP3 player, while downloading another MP3 file and playing Solitaire at the same time. Therefore, instead of setting the system performance level directly through Solitaire for example, it would be better if Solitaire notifies the operating system (OS) scheduler about predicted user delays and leave it up to the OS to globally schedule all the processes. The OS scheduler can make use of the predicted user delays to allocate time slots and performance levels to different processes in an energy-efficient fashion.

### C. *History-based model vs. psychological model*

The history-based model enjoys simplicity of implementation. No knowledge about the user interface content is required. It is also more flexible in making different tradeoffs between energy savings and lazy errors. On the other hand, the psychological model generates negligible lazy errors. However, it has to know the user interface content, which may be complex and dynamic. Moreover, the Hick-Hyman Law has limitations in modeling complex cognitive processes. The psychological model is more accurate for user interfaces that do not involve complicated cognitive processes. Therefore, different delay models should be used for different states to exploit the strengths of both the history-based and psychological models.

### *D. Improving prediction accuracy*

We showed in Section VI that the energy saving ratio is directly related to user delay prediction accuracy. The experimental results also demonstrated that more energy can be saved when delay can be predicted more accurately. To better utilize user delays for energy reduction, improving prediction accuracy will be important.

### *E. Adaptation*

Users may differ from each other in their perceptual, cognitive, and motor delays. The same user may improve his/her usage skills due to the learning process. Moreover, the environment or context may also change user behavior. Although the history-based delay model automatically adapts, the psychological model does not. It would be interesting to see how adaptation would improve prediction accuracy and increase energy savings.

## IX. Conclusions

Based on an analysis of its power characteristics, we found an interactive system spends most of its time and energy waiting for user response, and therefore the most effective way to reduce system energy consumption is to reduce the energy consumed during user delays.

We proposed a user interface based user delay prediction framework. In this framework, an STD is first obtained for the software to model the interaction between the system and user. User delays are then predicted based on different STD states. Within this framework, two prediction models were proposed. We theoretically showed how prediction errors are related to energy savings through DPM/DVS. The history-based model predicts the actual delay based on recent observations. Since it may overestimate the delay, it should only be used when performance level transition delay is not a large concern. In our experiments, it resulted in an average system energy saving of 20.7% with a relatively small percentage of serious lazy errors. We also showed that exploiting STD states yielded a better tradeoff between lazy errors and energy savings. The psychological model exploits the user interface information further and predicts the lower bound on user delays, *i.e.*, how long it takes the user to read, decide, and move. Our experiments show that an average of 21.9% system energy savings can be obtained with negligible serious lazy errors. We showed that the tradeoff between lazy errors and energy savings achieved by the psychological model is beyond the capacity of the history-based model.

Beside utilizing user delay predictions, we also showed how DPM/DVS can be simply combined with the user interface. An average of 17.6% system energy reduction can be easily achieved by inserting DPM/DVS code into the user interface without introducing user-noticeable delays. For applications more tolerant of system delays and with longer user delays, an average of 28.9% system energy reduction can be achieved.

## Acknowledgments

The authors would like to thank Yuanyuan Zhang and Pallav Gupta for participating as users in the usage trace collection phase.

## References

[1] Benchmarks for usage trace collection, *http://www.ee.princeton.edu/~cad/benchmarks.html.*

[2] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” *IEEE Trans. VLSI Systems*, vol. 8, no. 3, pp. 299–316, June 2000.

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

[4] R. P. Carver, *Reading Rate: A Review of Research and Theory*, Academic Press, Inc., San Diego, CA, 1990.

[5] I. Choi, H. Shim, and N. Chang, “Low-power color TFT LCD display for handheld embedded systems,” in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2002, pp. 112–117.

[6] P. M. Fitts, “The information capacity of human motor system in controlling the amplitude of movement,” *J. Experimental Psychology*, no. 47, pp. 381–391, 1954.

[7] K. Flautner, S. Reinhardt, and T. Mudge, “Automatic performance setting for dynamic voltage scaling,” in *Proc. Ann. Int. Conf. Mobile Computing and Networking*, July 2001, pp. 260–271.

[8] F. Gatti, A. Acquaviva, L. Benini, and B. Ricco, “Low power control techniques for TFT LCD displays,” in *Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, Oct. 2002, pp. 218–224.

[9] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. Neufeld, “Policies for dynamic clock scheduling,” in *Proc. Symp. Operating Systems Design & Implementation*, Oct. 2000, pp. 73–86.

[10] H. R. Hartson, A. C. Siochi, and D. Hix, “The UAN: A user-oriented representation for direct manipulation interface designs,” *ACM Trans. Information Systems*, vol. 8, no. 3, pp. 181–203, July 1990.

[11] W. E. Hick, “On the rate of gain of information,” *Quarterly J. Experimental Psychology*, no. 4, pp. 11–36, 1952.

[12] A. J. Hornof and D. E. Kieras, “Cognitive modeling reveals menu search is both random and systematic,” in *Proc. Conf. Human Factors in Computing Systems*, 1997, pp. 107–114.

[13] R. Hyman, “Stimulus information as a determinant of reaction time,” *J. Experimental Psychology*, no. 45, pp. 188–196, 1953.

[14] Intel StrongARM SA-1110 Microprocessor Developer’s Manual, *http://www.intel.com/design/strong/manuals/278240.htm.*

[15] N. K. Jha, “Low power system scheduling and synthesis,” in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2001, pp. 259–263.

[16] J. R. Lorch and A. J. Smith, “Improving dynamic voltage scaling algorithms with PACE,” in *Proc. ACM SIGMETRICS*, June 2001, pp. 50–61.

[17] I. S. MacKenzie, *Toward a Multidisciplinary Science of Human-Computer Interaction*, Morgan Kaufmann, San Francisco, CA, 2003, ch. II. Motor Behavior Models for Human-Computer Interaction, pp. 27–54.

[18] I. S. MacKenzie and W. Buxton, “Extending Fitts’ law to two-dimensional tasks,” in *Proc. Conf. Human Factors in Computing Systems*, 1992, pp. 219–226.

[19] T. Murata, “Petri nets – Properties, analysis, and applications,” *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[20] K. L. Norman, *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*, Alex Publishing Corporation, Norwood, NJ, 1991.

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

[22] J. Pouwelse, K. Langendoen, and H. Sips, “Energy priority scheduling for variable voltage processors,” in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2001, pp. 28–33.

[23] Qtopia, *http://www.trolltech.com/products/qtopia/.*

[24] R. W. Soukoreff and I. S. MacKenzie, “Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard,” *Behaviour & Information Technology*, no. 14, pp. 370–379, 1995.

[25] T. Šimunić, L. Benini, and G. De Micheli, “Event-driven power management of portable systems,” in *Proc. Int. Symp. System Synthesis*, Nov. 1999, pp. 18–23.

[26] M. A. Viredaz and D. A. Wallach, “Power evaluation of a handheld computer,” *IEEE Micro*, vol. 23, no. 1, pp. 66–74, Jan./Feb. 2003.

[27] A. Wasserman, “Extending state transition diagrams for the specification of human-computer interaction,” *IEEE Trans. Software Engineering*, vol. 11, no. 8, pp. 699–713, Aug. 1985.

[28] L. Zhong and N. K. Jha, “Graphical user interface energy characterization for handheld computers,” in *Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, Nov. 2003.