AURA: An Application and User Interaction Aware Middleware Framework for Energy Optimization in Mobile Devices

Brad K. Donohoo, Chris Ohlsen, and Sudeep Pasricha
bdonohoo@rams.colostate.edu, ohlsensc@rams.colostate.edu, sudeep@colostate.edu
Colorado State University, Department of Electrical and Computer Engineering
Fort Collins, CO, U.S.A

***Abstract* — Mobile battery-operated devices are becoming an essential instrument for business, communication, and social interaction. In addition to the demand for an acceptable level of performance and a comprehensive set of features, users often desire extended battery lifetime. In fact, limited battery lifetime is one of the biggest obstacles facing the current utility and future growth of increasingly sophisticated "smart" mobile devices. This paper proposes a novel application-aware and user-interaction aware energy optimization middleware framework (*AURA*) for pervasive mobile devices. *AURA* optimizes CPU and screen backlight energy consumption while maintaining a minimum acceptable level of performance. The proposed framework employs a novel Bayesian application classifier and management strategies based on Markov Decision Processes to achieve energy savings. Real-world user evaluation studies on a Google Android based HTC Dream smartphone running the *AURA* framework demonstrate promising results, with up to 24% energy savings compared to the baseline device manager, and up to 5× savings over prior work on CPU and backlight energy co-optimization.**

I. Introduction

Mobile smartphones and other portable battery operated embedded systems (PDAs, tablets) are pervasive computing devices that have emerged in recent years as essential instruments for communication, business, and social interactions. As of 2011, there are more than 5.3 billion mobile subscribers worldwide, with smartphone sales showing the strongest growth. Over 300,000 apps have been developed within the past three years across various mobile platforms. Popular mobile activities include web browsing, multimedia, games, e-mail, and social networking [1]. Overall, these trends suggest that mobile devices are now the new development and computing platform for the 21st century.

Performance, capabilities, and design are all primary considerations when purchasing a mobile device; however, battery lifetime is also a highly desirable attribute. Most portable devices today make use of lithium-ion polymer batteries, which have been used in electronics since the mid 1990's [2]. Although lithium-ion battery technology and capacity has improved over the years, it still cannot keep pace with the power consumption demands of today's mobile devices. Until a new battery technology is discovered, this key limiter has led to a strong research emphasis on battery lifetime extension, primarily using software optimizations [3]-[10].

It is important to note that outside of the obvious differences between portable mobile devices and a general PC – weight and size, form factor, computational capabilities, and robustness – a key difference can be found in the user interaction patterns and interfaces. Unlike a desktop or notebook PC in which a user typically interacts with applications using a pointer device or keyboard, applications on mobile devices most often receive user input through a touch screen or keypad events. Many times applications are interacted with for short durations throughout the day (e.g. few seconds or minutes instead of hours) and these patterns are often unique to each individual user. Significant differences in user interaction patterns make a general-purpose power management strategy unsuitable for mobile devices.

In this work, we present a novel application and user interaction aware energy management framework (*AURA)* for pervasive mobile devices, which takes advantage of user idle time between interaction events of the foreground application to optimize CPU and backlight energy consumption. In order to balance energy consumption and quality of service (QoS) requirements that are unique to each individual user, *AURA* makes use of a Bayesian application classifier to dynamically classify applications based on user interaction. Once an application is classified, *AURA* utilizes Markov Decision Process (MDP) based power management algorithms to adjust processor frequency and screen backlight levels to reduce system energy consumption between user interaction events. Overall, we make the following novel contributions:

- We conduct usage studies with real users and develop a Bayesian application classifier tool to categorize mobile applications based on user interaction activity
- We develop an integrated MDP-based application and user interaction-aware energy management framework that adapts CPU and backlight levels in a mobile device to balance energy consumption and user QoS
- We characterize backlight and CPU power dissipation on an Android OS based HTC Dream mobile architecture
- We implement our framework as middleware running on the HTC Dream smartphone and demonstrate real energy savings on commercial apps running on the device

Real-world user evaluation studies with the Google Android based HTC Dream mobile device running the *AURA* framework demonstrate promising results, with up to 24% energy savings compared to the baseline device manager; and up to 5× savings over the best known prior work on CPU and backlight energy co-optimization, with negligible impact on user quality of service.

II. *AURA* Energy Management Framework

In this section, we present details of the *AURA* framework. In Section II.A we first describe the fundamental observations that lay the foundation for energy savings in mobile devices. Section II.B presents results of field studies involving users interacting with apps on mobile devices. Section II.C gives a high level overview of the *AURA* middleware framework. Subsequent sections elaborate on the major components of the framework.

A. Fundamental User-Device Interaction Mechanisms

Here we explain the underlying concepts stemming from the psychology of user-device interactions that drive the CPU and backlight energy optimizations in the *AURA* framework.

There are three basic processes involved in a user's response to any interactive system [11], such as a smartphone or personal

978-1-4577-1954-7/11/$26.00 ©2011 IEEE

computer. During the *perceptual* process, the user senses input from the physical world. During the *cognitive* process, the user decides on a course of action based on the input. Finally, during the *motor* process, the user executes the action by mechanical movements. These three processes are consecutive in time and can be characterized by an *interaction slack*. For instance, when interacting with an app on a mobile device, the time between when a user interacts with an app (e.g., touching a button) and when a response to that interaction is perceptible to the user (e.g., the button changes color) is the perceptual slack; the period after the system response during which the user comprehends the response represents the cognitive slack; and finally the manual process of the next interaction (e.g. moving finger to touch the screen) involves a motor slack. Before the user physically touches the input peripherals, the system is idle during the cumulative slack period, for hundreds to possibly many thousands of milliseconds. During this time, if the CPU frequency can be reduced, energy may be saved without affecting user QoS. **CPU energy** optimizations in *AURA* exploit this inherent slack that arises whenever a user interacts with a mobile device.

Another interesting phenomenon that can open up new possibilities for energy optimizations is the notion of *change blindness* [12] as revealed by researchers in human psychology and perception. Change blindness refers to the inability of humans to notice large changes in their environments, especially if changes occur in small increments. Many studies have confirmed this limitation of human perception [13] – a majority of observers in one study failed to observe when a building in a photograph gradually disappeared over the course of 12 seconds; in another study, gradual color changes over time to an oil painting went undetected by a majority of subjects but disruptive changes such as the sudden addition of an object were easily detected. By gradually reducing screen brightness over time under certain scenarios, energy consumption in mobile devices may thus be reduced without causing user dissatisfaction. This principle is exploited by *AURA* to optimize screen **backlight energy**.

B. User-Device Interaction Field Study

To understand how users interact with mobile applications in the real world, we conducted a field study with users running a variety of apps on a popular smartphone. It is well-known that apps running on smart mobile devices can invoke vastly different interaction behaviors from users – whereas some applications require constant interaction with the user, others may experience bursts of user interaction followed by long idle periods. These unique interaction patterns make conventional general-purpose energy management strategies ineffective on mobile devices. We hoped to uncover trends from the field studies that would potentially guide scenario-specific energy optimizations.

We selected 18 Android apps from the following categories: games, communication, multimedia, shopping, personal/business, travel/local, and social networking. A custom background interaction logger app was developed to record interaction traces for users running each of these apps. The study involved five different users interacting with the apps over the course of a day on an Android OS based HTC Dream smartphone. The users selected for the study spanned the proficiency spectrum: user 3 for instance mainly used his smartphone for phone calls, and was not familiar with many of the selected applications, whereas user 2 relied on her smartphone for a variety of tasks, and was quite comfortable while interacting with the chosen apps.

Table 1 shows a summary of the interaction logged for the users for each of the selected apps. The patterns in the table are classified based on interaction frequency (low, medium, high). An obvious observation we can immediately make from these results is that although some apps belong to the same category (e.g., *Sudoku* and *Jewels* are both "games") they possess vastly different interaction characteristics. It is also interesting to see that user-device interactions for some apps are very **application-specific**. For example, *Jewels*, a fast-paced gaming application, is always classified as high-interaction, and *Music* is always classified as low-interaction because users usually select a song then cease interaction. On the other hand, some classifications are noticeably **user-specific**, meaning that they may be classified quite differently based on the type of user interacting with them. Interaction results for *Minesweeper*, *Gallery*, and *News & Weather* all illustrate this idea.

We can thus conclude that users interact with devices in a user-specific and application-specific manner. Any framework that would attempt to exploit interaction slack and change blindness to save energy (as discussed in the previous section) must tailor its behavior uniquely across users and applications.

Table 1: App interaction classification

APPLICATION	CLASSIFICATION				
	User 1	User 2	User 3	User 4	User 5
1 Market (com.android.vending)	High	High	Med	High	Med
2 Gallery (com.cooliris.media)	Med	High	Low	Med	High
3 Jewels (org.mhgames.jewels)	High	High	High	High	High
4 Wordfeud FREE (com.hbwares.wordfeud.free)	Low	Low	Low	Low	Low
5 Facebook (com.facebook.katana)	High	High	Med	High	Med
6 Gmail (com.google.android.gm)	Med	High	Med	Med	Med
7 SMS(com.android.mms)	High	High	High	High	High
8 Browser (com.android.browser)	High	High	High	Med	Med
9 News & Weather (com.google.android.apps.genie.geniewidget)	High	Med	Med	High	Low
10 YouTube (com.google.android.youtube)	Low	Low	Low	Low	Low
11 Calculator (com.android.calculator2)	High	High	Med	High	High
12 Twitter (com.android.twitter)	Med	High	Med	High	High
13 Calendar (com.android.calendar)	Low	Low	Med	Med	Low
14 Google Maps (com.google.android.apps.maps)	Med	High	Med	Med	Med
15 Sudoku (cz.romario.opensudoku)	Low	Med	Low	Low	Low
16 Music (com.android.music)	Low	Low	Low	Low	Low
17 Solitaire (com.softick.android.solitaire.klondike)	High	High	High	High	High
18 Minesweeper (artfulbits.aiMinesweeper)	Med	High	Med	High	High

C. AURA Middleware Framework Overview

We now present a high level overview of the *AURA* energy optimization framework for mobile devices. Figure 1 shows the *AURA* framework that is implemented at the middleware level in the software stack of the Android OS [14], and runs in the background as an Android Service. As the user switches between applications, the *AURA* **runtime monitor** will receive a "Global Focus Event," and update any relevant session information of the previous foreground application to an **app profile database**, stored in memory. The runtime monitor will then search the database for the current foreground application. If the application exists in the database and is classified, the **MDP power manager** will make use of the application's learned classification and user interaction pattern statistics (e.g., mean and standard deviation) to invoke a user-specific power management strategy that is optimized to the application. The power manager directly accesses the mobile device's hardware components to change screen backlight and CPU frequency. If the application does not exist in the database, or the application exists in the database but is not classified, *AURA* will make use of the user interaction events ("Global Touch" and "Global Key" events) to dynamically classify the application using the **Bayesian application classifier**. Following classification, the MDP power manager will run an appropriate power management strategy on the next invocation. An **event emulator** was also integrated into *AURA* to simulate user interaction events for validation purposes.

D. Runtime Monitor

The runtime monitor is an event-driven module responsible for monitoring user interactions with the mobile device. The Android

OS makes use of public callback methods, known as event listeners, which allow applications to receive and handle user interactions (e.g., touch and key events) with items displayed on the user interface (UI) [14]. Unfortunately, there is no API to allow applications to receive "global" UI events, which are often consumed by application code or the current foreground application. In order for the runtime monitor to receive global UI events, a set of custom event broadcasts were created that are sent when any touch or key event occurs, when an application gains focus, and when the phone configuration changes (e.g., hard keyboard is opened or closed). The runtime monitor is only invoked whenever such custom global UI events are broadcast. The module keeps tracking variables for each current foreground application, the class of user interaction events, and the phone state, which are updated on receiving global UI events. When a new application gains foreground focus, the runtime monitor will save session information from the previous foreground application (e.g. classification, user interaction stats) to the app profile database and retrieve any stored application information from the database for the new foreground application. If the new foreground application is classified, then the runtime monitor will invoke the MDP power manager, otherwise, it will invoke the Bayesian classifier to dynamically classify the application as it receives user interaction events.

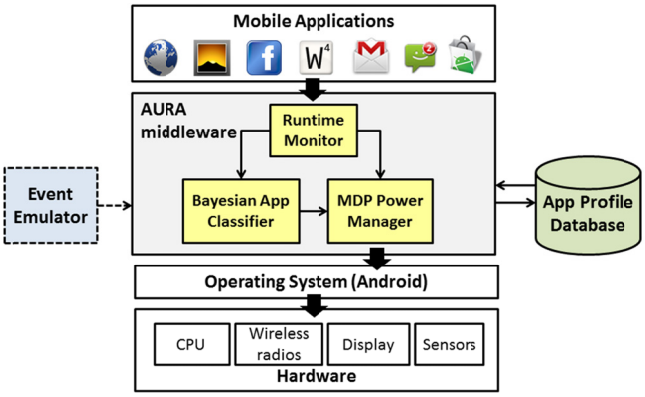

Figure 1: *AURA* energy optimization middleware framework

E. Bayesian Application Classifier

Bayesian learning [15] is a form of supervised learning that involves using evidence or observations along with prior outcome probabilities to calculate the probability of an outcome. More specifically, the probability that a particular hypothesis is true, given some observed evidence (the *posterior probability* of the hypothesis), comes from a combination of the *prior probability* of the hypothesis and the compatibility of the observed evidence with the hypothesis (the *likelihood* of the evidence). *AURA* uses a form of Bayesian learning based on the classification method executed by Jung et al. [16]. This method of classification involves mapping a set of *input features*, $X = \{x_1, x_2, \ldots, x_n\}$, to a set of *output measures,* $Y = \{y_1, y_2, \ldots, y_n\}$. In our work, the input features are the mean (x_1) and standard deviation (x_2) of the times between user interaction (e.g., touch) events. We create a simple tripartite categorization of input features into *low*, *medium*, and *high* classes based on corresponding values of mean and standard deviation. We define a single output measure: the resulting application classification (y). The pre-defined application classes are *low-interaction*, *medium-interaction*, or *high-interaction*. Thus the classifier ultimately allows us to get a characterization of user and application specific interaction intensity. Learning for our classifier is accomplished through creation of a training set of size γ_{TS} for each application. Finding a consistent mapping from input features to an output measure using this training set enables the classifier to accurately predict the class of an application.

The classification task consists of calculating *posterior probabilities* for each class, based on the *prior probability*, *class likelihood,* and *evidence*, then choosing the class with the highest posterior probability. The posterior probabilities are obtained using the following equation:

$$posterior\ probability = \frac{prior\ probability \times likelihood}{evidence}$$

$$\downarrow$$

$$Prob(y = c|x_1, x_2) = \frac{Prob(x_1, x_2|y = c) \times Prob(y = c)}{Prob(x_1, x_2)}$$

The denominator $Prob(x_1, x_2)$, or the evidence, is the marginal probability that an observation X is seen under all possible hypotheses, and is irrelevant for decision making since it is the same for every class assignment [15]. $Prob(y = c)$, the class likelihood, is the prior probability of the hypothesis that the application class y is c. This is easily calculated from the training set. $Prob(x_1, x_2|y = c)$ is the per-input-feature conditional probability of seeing the input feature vector X given that the application class y is c. From Jung et al. [16] we have:

$$Prob(x_1, x_2|y = c) = Prob(x_1|y = c) \times Prob(x_2|y = c)$$

Thus, the posterior probability of an application class may be computed as follows:

$$Prob(y = c) = Prob(x_1, x_2|y = c) \times Prob(y = c)$$

There may be some cases in which input features do not occur together with an output measure due to an insufficient number of data points in the training set, resulting in a zero conditional probability. To deal with this, we eliminate them by smoothing:

$$Prob(x_i|y = c) = \frac{freq(x_i, y = c) + \lambda}{freq(y = c) + \lambda n_x}$$

where λ is a smoothing constant ($\lambda > 0$), and n_x is the number of different attributes of x_i that have been observed.

Table 2: Example Training Set

Input Features		Output Measure
Mean	**Std. Dev.**	**Classification**
high	low	high-interaction
high	high	med-interaction
med	med	med-interaction
med	low	med-interaction
low	high	low-interaction
med	high	med-interaction

Consider an example of dynamic classification based on the training set shown in Table 2. When looking at this table, it should be noted that x_1=*high* denotes a *high-interaction* mean value, thus a smaller mean gap between events. Let the bottom row be a new input feature (x_1=*med*, x_2=*high*) which was not in the training set, that is presented to the classifier. If $\lambda = 1$, this new input is classified as follows. For the hypothesis *y=high-interaction*, the posterior probability is: $Prob(x_1$=*med*,x_2=*high*$|y$=*high-interaction*$)$=(1/6)×(1/6)×(1/5)=1/180 because $Prob(x_1$=*med*$|y$=*high-interaction*$)$=1/6 and $Prob(x_2$=*high*$|y$=*high-interaction*$)$ =1/6. Similarly, for the hypothesis *y=med-interaction*, posterior probability is: $Prob(x_1$=*med*,x_2=*high*$|y$=*med-interaction*$)$=(3/8)× (1/4)×(3/5)=9/160; and for hypothesis *y=low-interaction*,

posterior probability is: $Prob(x_1=med,x_2=high|y=low\text{-}interaction) = (1/6)\times(1/3)\times(1/5)=1/90$. The output measure of the new input feature is thus *med-interaction*.

An application is classified in this manner on each interaction event. Upon being assigned the same classification *m* (a user-defined parameter) number of times, the application is marked as classified, and an appropriate power management strategy will be run on its next invocation.

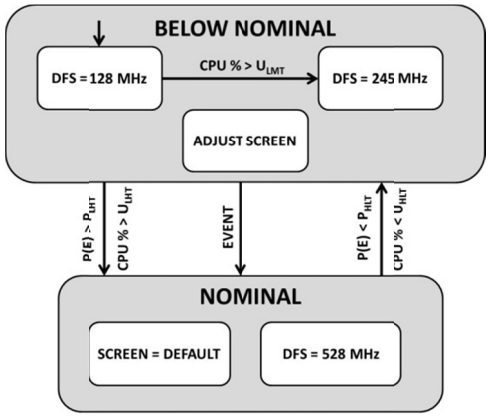

Figure 2: Control Algorithms State Flow Diagram

F. MDP Power Manager

Markov Decision Processes (MDP) [17] are discrete time stochastic control processes that are widely used as decision-making models for systems in which outcomes are partly random and partly controlled. MDPs consist of a four-tuple (S, A, P_A, $R_{S,A}$), where S is a finite number of states, A is a finite set of actions available from each state, P_A is the probability that action a in state s at time t will lead to state s^* at time $t + 1$, and $R_{S,A}$ is a reward from transitioning to state s^* from state s.

As part of the *AURA* framework, we develop three MDP-based power management algorithms to transition between a nominal state and below nominal state at discrete time intervals based on probabilities of user interaction, processor demands, and application classification. All three algorithms make use of the state flow diagram illustrated in Figure 2. When an application is classified and gains foreground focus, the classification and learned user event statistics are used to set algorithm control parameters (e.g., evaluation interval τ, screen backlight adjustment levels Δ_S, probability thresholds P_T) and speculate on the probability of a user event within the next evaluation interval. When the probability of an event is below a given high-to-low threshold (P_{HLT} in Figure 2) and processor utilization is below a given utilization threshold (U_{HLT} in Figure 2), the state transitions to *Below Nominal* and the processor frequency level is adjusted to the lowest supported frequency. While the state is *Below Nominal*, the screen brightness is gradually adjusted down at each evaluation interval, taking advantage of the concept of *change blindness*. A transition to the mid-level frequency while in the *Below Nominal* state will also occur if processor utilization reaches a low-med utilization threshold (U_{LMT} in Figure 2) or an immediate transition back to *Nominal* state will occur if processor utilization exceeds an even higher utilization threshold (U_{LHT} in Figure 2). If processor utilization is not a dominating factor then a transition back to *Nominal* will occur as the probability of a user event approaches a given low-to-high threshold (P_{LHT} in Figure 2) or any user event occurs, in which the nominal processor frequency and screen backlight level are set back to user-specified defaults.

Our baseline algorithm, *NORM*, makes use of a Gaussian distribution to predict user events based on learned classification and stored user-interaction statistics. Two derivatives of *NORM* were created to dynamically adapt to real-time user-interaction during each classified application invocation. The *E-ADAPT* variant is event-driven and uses the most recent W_E user events to predict future interaction events. The *T-ADAPT* variant makes use of a moving average window of size W_T, to dynamically track recent user interaction within a temporal context.

Table 3: Algorithm Control Parameters

Parameter	Dependencies	Description
Evaluation Interval (τ)	Application Classification	Discrete time interval in which to re-evaluate the system state and decide appropriate action(s)
Event Probability Threshold (P_T)	Application Classification	Threshold to determine state transitions based on user-event prediction
Processor Utilization Threshold (U_T)	Static	Conditional threshold to determine state transitions based on CPU demands
Backlight Adjustment Levels (Δ_S)	COV†	Screen backlight adjustment levels while in state *Below Nominal*
Event Window (W_E)	E-ADAPT Only	Number of most recent user events to use for dynamic adaptation
Time Window (W_T)	T-ADAPT Only	Size of moving average window to use for dynamic adaptation

†COV ≡ Coefficient of Variance

Table 3 above lists several key algorithm control parameters that can be used to customize the algorithms to each individual user and application. For example, the evaluation interval is determined based on the application classification. High interaction applications will require more frequent evaluation, while low interaction applications can use a larger evaluation interval. Event probability thresholds are also set based on application classification and coefficient of variance, which gives a measure of the variability of the learned user interaction pattern. For example, low interaction applications with low variability will use high event probability thresholds to bias towards energy savings. For applications with high variability in user-interaction patterns, event probability thresholds will be slightly biased towards QoS. Processor utilization thresholds are statically set at design time; however, could also be set dynamically if finer-grained control is required. Screen backlight levels are set based on application classification, where high interaction applications are again biased towards QoS and low interaction applications are biased towards energy savings. It is important to note that screen backlight levels are also limited by a QoS-driven lower bound of 40%. For the *E-ADAPT* variant, a larger W_E value can be used to get a global view of the user-interaction pattern or a smaller W_E value can be used for a more local view of the user-interaction pattern. A similar argument can be made for the *T-ADAPT* variant which dynamically adapts the user-event statistics based on a moving average window of size W_T.

Figure 3 illustrates the basic working of the MDP-based power management algorithms. At Point 1, the probability of a user event reaches the low-to-high probability threshold and the state transitions back to *Nominal*. After an event occurs at Point 2, the probability of a user event falls below the high-to-low threshold, Point 3, and the state transitions to the *Below Nominal* state. While in the below nominal state, the screen backlight gradually changes during this predicted user idle time. At Point 4, the probability of a user event rises above the low-to-med threshold and the processor frequency is adjusted accordingly. At Point 5,

the probability of a user-event reaches the low-to-high threshold and the state transitions back to *Nominal*. Point 6 shows a misprediction, in which a user event occurs while in the *Below Nominal* state. This causes the state to transition back to *Nominal* again. Mispredictions are used as a coarse QoS metric to evaluate the effectiveness of the algorithms and ensure energy savings are not diminishing overall QoS.

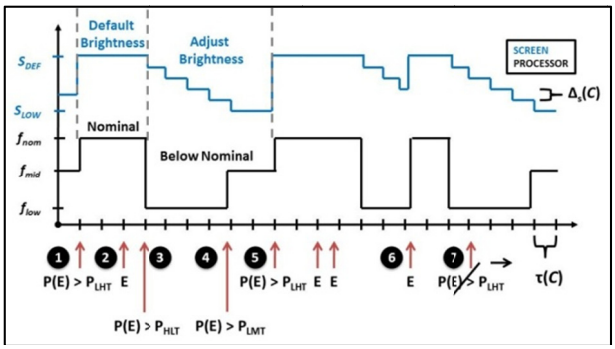

Figure 3: Control Algorithms Timeline Illustration

III. Device Power Modeling

A. Overview

In order to quantify the energy-effectiveness of the proposed framework, power analysis was performed on the Android based HTC Dream smartphone, with the goal of creating power models for the CPU and backlight. We use a variant of Android OS 2.2, (Froyo) for the HTC Dream and the Android SDK Revision 11. The Google Android platform is comprised of a slightly modified 2.6.35.9 Linux kernel, and incorporates the Dalvik Virtual Machine (VM), a variant of Java implemented by Google. All user-space applications are Dalvik executables that run in an instance of the Dalvik VM. We built our power estimation models using real power measurements on the HTC Dream device. We instrumented the contact between the smartphone and the battery, and measured current using the Monsoon Solutions power monitor [18]. The monitor connects to a PC running the Monsoon Solutions power tool software that allows real-time current/power measurements over time. We developed a custom test app that was capable of switching dynamic frequency scaling (DFS) levels and screen backlight levels over time. The corresponding power traces from the Monsoon Power Tool were then used to obtain power measurements for each DFS frequency and screen brightness level.

B. CPU DFS Power Model

Dynamic frequency scaling (DFS) is a standard technique in computing systems to conserve power by dynamically changing frequency at runtime when the system is idle or under-utilized. Linux-based systems, such as Android, provide a DFS kernel subsystem (*cpufreq*) that is a generic framework for managing the processor operation. The *cpufreq* subsystem supports a variety of DFS drivers, many of which are configurable [19]. The *userspace* driver, which is used in this work, allows a user-space program to manually control DFS levels based on a user-defined set of rules.

The HTC Dream smartphone consists of a Qualcomm® MSM7201A processor. This processor is nominally clocked at 528 MHz; however, it supports 11 different frequency levels ranging from 128 MHz up to 614 MHz. In order to get a power model for the DFS levels, the test app was used to manually switch frequency domains. Given the difficulty of controlling CPU utilization explicitly, an intensive computation loop, followed by a wait function was used to toggle processor utilization between 0 and 100% for each frequency level, and the power consumption was recorded respectively. Based on the observed measurements, linearity was assumed for CPU utilization that lies between 0 and 100 % for each frequency. Linear regression [20] was used to determine appropriate coefficients and develop our CPU power estimation model with units of *mW/% utilization*:

$$P_{CPU} = 0.0024 \times freq\ (MHz) + 3.0511$$

C. Screen Backlight Power Model

Screen power consumption was also measured using the Monsoon Power Tool and our custom test app, which allowed static and dynamic control of screen backlight levels. Screen backlight levels were incremented in increments of 10% and relative average power measurements were recorded with the Monsoon Power Tool. Figure 4 shows the instantaneous power measured using the tool as the backlight level is ramped up. Linear regression was again used to obtain a relative power model with units of *mW*:

$$P_{SCREEN} = 596.8 \times \psi$$

where ψ is the backlight level ($0.0 \leq \psi \leq 1.0$).

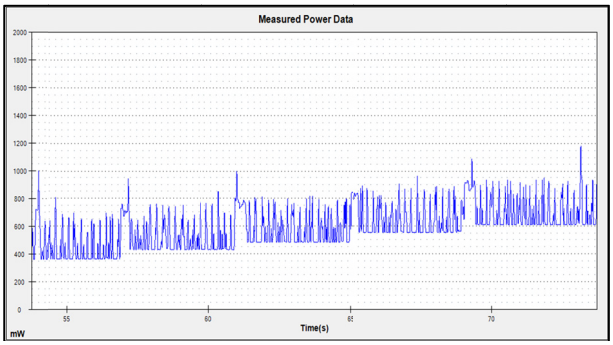

Figure 4: Monsoon Power Monitor Screen Power

IV. Experimental Methodology

We implemented our *AURA* framework on a real HTC Dream smartphone. The CPU and backlight power models described above were integrated into the *AURA* framework, to guide our MDP-based power management strategies and also to quantify any resulting energy savings. To test the effectiveness of the *AURA* framework, we obtained stimuli from two sources:

- **Event emulator:** This module emulates various key and touch based user interaction event patterns, including: *(i) Constant* – sends events at a user-defined constant rate; *(ii) Stochastic* – sends events using a Gaussian distribution probability with a user-defined mean and standard deviation; *(iii) Random* – sends events using a pseudo-random uniform distribution; and *(iv) High Burst Long Pause* – sends a quick burst of 20 events, followed by a long 10 second pause.
- **Real users:** We involved five users who interacted with applications as they would normally, generating user-specific interaction events in the process.

The following algorithm control parameters were assumed in our implementation: for the Bayesian classifier, $\lambda = 1$ and $\gamma_{TS} = 50$; and for the MDP algorithms, $W_E = 6$ and $W_T = 6$. For the state

flow transitions, P_{LHT} = 0.5/0.6/0.7 (H/M/L Classification), P_{HLT} = 0.3/0.4/0.5 (H/M/L Classification), U_{LHT} = 0.8, U_{LMT} = 0.6, U_{HLT} = 0.5, and Δ_S = 0.05/0.1/0.15 (H/M/L COV).

V. Results

A. Event Emulation Results

In our first study, the event emulator was used to imitate four different user-interaction patterns, to contrast the effectiveness of our three MDP-based power management algorithms. We were primarily interested in the energy savings from our algorithms compared to nominal frequency and default screen backlight settings on the HTC Dream device. To evaluate the level of QoS being offered, a *successful prediction rate* metric was defined. The successful prediction rate metric keeps track of the number of events that occurred while in *Below Nominal* state. A low successful prediction rate is indicative of diminished QoS. QoS is difficult to quantify on a user level – two users may perceive a common event very differently. Therefore, the successful prediction rate is merely an attempt at measuring user satisfaction. The algorithm control parameters used in our experiments were biased towards maintaining fewer mispredictions, but could be adjusted to allow more.

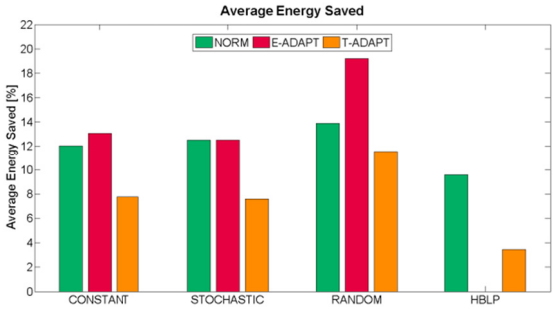

Figure 5: Average Energy Saved for MDP Algorithms

Figure 5 shows the energy savings achieved by the three algorithms for the four emulated interaction patterns on the x-axis. It can be seen that *E-ADAPT* offers the best overall energy savings (up to 20%) due to the responsive event-triggered nature of its dynamic adaptation. The algorithm only adapts on instances of user events and does not require constant adaptation during each evaluation interval. This also reveals why *E-ADAPT* performs poorly in conditions like *HBLP* - the quick events during the burst are stored in the *E-ADAPT* window as they occur and used to calculate the mean for prediction, but the long pause is not taken into account until the event *after* the long pause occurs. On the other hand, *T-ADAPT* offers lower energy savings due to its time-based nature. Whereas *E-ADAPT* must wait for an event occurrence in order to adjust its prediction, *T-ADAPT* adjusts on each evaluation interval. This form of adaptation is slightly more computationally intensive. Additionally, if the state is *Nominal*, it will not drop to *Below Nominal* under *T-ADAPT* unless CPU utilization is below U_{HLT} (even if the probability of an event is very low). Because *T-ADAPT* shifts its window and does computations on a regular interval instead of waiting for an event to occur, the CPU utilization is high more often, thus the state transitions to *Below Nominal* less often.

Figure 6 shows average successful prediction rates for each algorithm for with the event emulator patterns. It is clear that all three algorithms have high successful prediction rates, validating the robustness of our chosen classification and power management algorithms. A higher successful prediction rate, however, does not necessarily imply higher energy savings for the following reason: an event may be predicted to occur much earlier than the actual time of its occurrence and the prediction will still be considered successful. For example, if the state is *Below Nominal* and the probability of an event occurrence exceeds P_{LHT}, the state will switch to *Nominal* and remain there until an event occurs and the probability of an event occurrence is low again. Therefore, if an event does not occur quickly, less energy will be saved because the *Nominal* state will be prolonged (but the prediction will still be considered successful). This is the reason why although *T-ADAPT* offers the best successful prediction rates due to its superior temporal locality characteristics, it offers lower energy savings than *NORM* and *E-ADAPT*. It should be noted that in our experiments, the algorithm control parameters were biased towards high successful prediction rates, thus maintaining a high minimum level of user QoS for all three algorithms. The control parameters could potentially be adjusted to accept lower successful prediction rates in order to offer even higher energy savings.

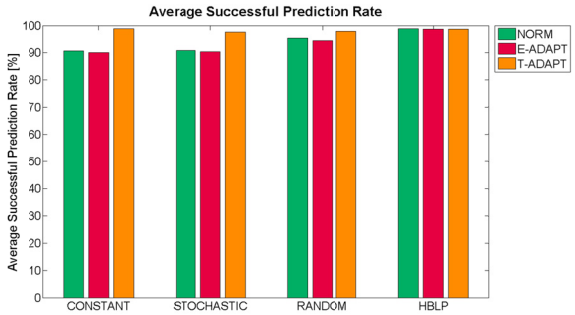

Figure 6: Avg. Successful Prediction Rate for MDP Algorithms

B. User Study Results

A second study was conducted, this time focusing on eight common Android apps and five real users using these apps on the HTC Dream device. The users were asked to interact with the different applications over the course of several sessions during a day, with different energy saving algorithms enabled in each session. In addition to our three MDP-based power management algorithms running as part of the *AURA* middleware framework, we implemented the framework proposed by Shye et al. [3] which also aims to improve CPU and backlight energy consumption for mobile devices by using change blindness to gradually ramp down both CPU frequency and screen backlight levels over time.

Figure 7 compares the average energy savings of the technique proposed by Shye et al. (*CHBL*) with our three MDP-based power management techniques for the eight different applications, across all users. It can be seen that *CHBL* offers fairly consistent (but low) energy savings across all applications. The consistency can be explained by noting that the change blindness optimization employed in *CHBL* is independent of user interaction patterns and application type, instead using constant time triggered scaling of the CPU frequency and backlight levels. In contrast, our user-aware and application-aware algorithms (specifically *E-ADAPT*) offer higher energy savings because they can dynamically adapt to the user interaction patterns and take full advantage of user idle time. For instance, the *E-ADAPT* algorithm saves up to 4× and 5× more energy compared to *CHBL* for the *Gmail* and *WordFeud* applications, respectively. In some cases, for instance the high interaction *Jewels* application, interaction slack is too small to be exploited by our algorithms. While *CHBL* offers higher energy savings for *Jewels* due to its lack of user-awareness and

application-awareness during scaling, it comes at a cost: noticeable QoS degradation issues for the user. Overall, our proposed *AURA* framework with the *E-ADAPT* scheme enables as much as 24% energy savings over the default device settings for frequency and screen backlight without noticeably degrading user QoS, and improves upon prior work (*CHBL* [3]) by as much as 5× for some applications.

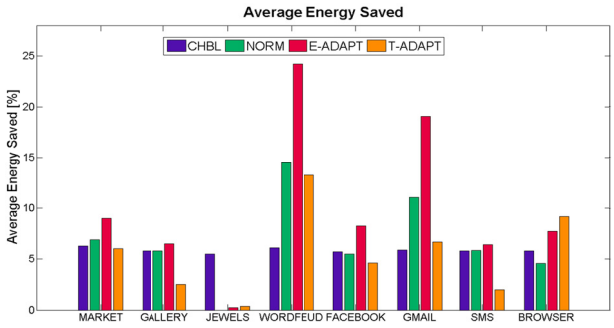

Figure 7: Average Energy Saved for Real Applications

VI. Related Work

A significant amount of work has been done in the area of energy optimization for mobile devices. Much of this work focuses on optimizing energy consumed by the device's wireless interfaces (e.g., WiFi, 3G/EDGE networks). In [5] an energy-aware handoff algorithm is proposed based on the energy consumption of UMTS (3G) and WiFi. The energy costs for transmitting data over different wireless interfaces is explored in [6], with the goal of balancing energy and delay during data transfers. Other work [7]-[9] focuses on energy-efficient location-sensing schemes aiming to reduce high battery drain caused by GPS usage. A framework for mobile sensing that efficiently recognizes contextual user states is proposed in [10]. Our work on optimizing the backlight and CPU energy is complementary to these works. While we do not optimize energy consumed by the location sensor and wireless interface, *AURA* can potentially be implemented alongside other strategies that do.

There have been some efforts to optimize CPU and backlight energy consumption in recent years. Shye et al. [3] in particular make several important observations that are relevant to this work: (1) the screen and the CPU are the two largest power consuming components, and (2) users are generally more satisfied with a scheme that gradually reduces screen brightness rather than one that changes abruptly. Based on the second observation, the authors implement a scheme that utilizes *change blindness* by slowly reducing screen brightness and CPU frequency over time. However, their approach does not consider application-aware and user-aware scaling like *AURA* does, because of which their approach tends to be overly conservative at times, and at other times can often detrimentally impacts user QoS. Bi et al. propose a user interaction-aware DFS approach in [4] much like *AURA*. However, their approach is directed towards stationary desktop systems, and not towards the touch-based mobile devices that are becoming more prevalent today. Their proposed approach is also different in that it does not consider backlight optimization, and uses a simple user-aware but application-unaware history predictor to set CPU frequency levels based on demand.

VII. Conclusions

In this work we proposed *AURA,* an application and user interaction aware energy optimization framework for pervasive mobile devices. As part of *AURA,* we introduced a novel method for classifying applications based on user interaction patterns using Bayesian classification, and three Markov Decision Process (MDP) based power management algorithms. We evaluated the framework on both emulated and real user interaction patterns, and found that it can achieve average energy savings up to 24% depending on the application, without perceptible impact on user QoS. When compared with prior work on CPU and backlight power savings in mobile devices [3], *AURA* can achieve as much as 5× their energy savings. Our ongoing work is focusing on developing better prediction techniques and battery models for quantifying lifetime improvement.

VIII. References

[1] MobiThinking, "Global mobile statistics 2011," http://mobithinking.com, April 2011.

[2] Micro Power White Paper, "Using Lithium Polymer Batteries in Mobile Devices," http://www.micro-power.com/userfiles/file/wp_polymer_final-1274743697.pdf [Online].

[3] A. Shye, B. Scholbrock, G. Memik, "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures," In *MICRO-42 '09*, pp. 168-178, Jan. 2009.

[4] M. Bi, I. Crk, C. Gniady, "IADVS: On-Demand Performance for Interactive Applications," In *HPCA '10*, pp. 1-10, Apr. 2010.

[5] H. Petander, "Energy-Aware Network Selection Using Traffic Estimation," In *MICNET '09*, pp. 55-60, Sept. 2009.

[6] M. Ra, et al., "Energy-Delay Tradeoffs in Smartphone Applications," In *MOBISYS '10*, pp. 255-270, Jun. 2010.

[7] I. Constandache, et al., "EnLoc: Energy-Efficient Localization for Mobile Phones," In *INFOCOM '09*, pp. 19-25, Jun. 2009.

[8] K. Lin, et al., "Energy-accuracy trade-off for continuous mobile device location," In *MOBISYS*, pp. 285-298. Jun 2010.

[9] Z. Zhuang, K. Kim, J. P. Singh, "Improving Energy Efficiency of Location Sensing on Smartphones," In *MOBISYS*, pp. 315-330,. 2010.

[10] Y. Wang, et al., "A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition," In *MOBISYS '09*, pp. 179-192, Jun. 2009.

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

[12] D. J. Simons, S. L. Franconeri, and R. L. Reimer, "Change blindness in the absence of a visual disruption", In *Perception*, 29:1143–1154, 2000.

[13] D. J. Simons, and R. A. Rensink, "Change blindness: Past, present, and future", Trends in Cognitive Sciences, Vol.9 No.1 January 2005.

[14] Google Android, official website, http://www.android.com

[15] E. Alpaydin, "Introduction to Machine Learning," 2nd ed. Massachusetts: The MIT Press, 2010.

[16] H. Jung, M. Pedram, "Improving the Efficiency of Power Management Techniques by Using Bayesian Classification," In *ISQED '08*, pp. 178-183, Mar. 2008.

[17] T. L. Cheung, K. Okomoto, F. Maker, X. Liu, V. Akella, "Markov Decision Process (MDP) Framework for Optimizing Software on Mobile Phones," In *EMSOFT '09*, pp. 11-20, Oct. 2009.

[18] Monsoon Solutions Inc., official website, http://www.msoon.com/LabEquipment/PowerMonitor, 2008.

[19] D. Brodowski et al., "Linux CPUFreq – CPUFreq Governors. Linux Documentation," http://mjmwired.net/kernel/Documentation/cpu-freq, March 2011.

[20] J.J. Faraway, "Linear Models with R", *CRC Press,* 2004

[21] HTC, "HTC G1 Overview," http://www.htc.com/www/product/g1/specification.html

[22] W. Liang, P. Lai, "Design and Implementation of a Critical Speed-based DFS Mechanism for the Android Operating System," In *EMS '10*, pp. 1-6, Sept. 2010.

[23] S. P. Tarzia, P. A. Dinda, R. P. Dick, G. Memik, "Display Power Management Policies in Practice," In *ICAC '10*, pp. 51-60, Jun. 2010.

[24] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, A. Rice, "Exhausting Battery Statistics: Understanding the energy demands on mobile handsets," In *MOBIHELD '10*, pp. 9-14, Aug. 2010.