User-centric Joint Power and Thermal Management for Smartphones

Pietro Mercati and Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California San Diego
San Diego, CA, USA
Email: {pimercat, tajana}@eng.ucsd.edu

Vinay Hanumaiah, Jitendra Kulkarni, and Simon Bloch
Advanced Systems Engineering Lab
Samsung Research America
San Jose, California, USA 95134
Email: {vinay, j.kulkarni, s.bloch}@samsung.com

***Abstract*—Excessive power consumption and high temperatures are some of the key factors limiting battery charge life and performance of current mobile devices or smartphones. Existing power management solutions do not account for user-experience, which can impact user satisfaction. In this paper we propose a joint power and thermal management solution, which takes a proactive approach in reducing energy consumption while providing expected user-experience. The proposed technique modulates the operating conditions based on users application preferences and exploits the "change blindness" effect to reduce display power consumption. Another important aspect of our implementation is that it does not require any restructuring of the underlying operating system. A novel thermal model of the entire smartphone was built with the purpose of monitoring and controlling the operating conditions to keep the device temperatures within safe operating ranges. Our ready-to-use management technique has been implemented on Google Nexus 5 and has been demonstrated to achieve a 46% application-specific savings on power consumption and up to 35% savings in power consumption at the device level. The mean temperature estimation error is 1.17°C.**

I. Introduction

Modern smartphones and tablets are composed of many heterogeneous subsystems (CPU, GPU, Display, Modem, 4G, WiFi, etc.), which are specialized to execute a wide variety of applications from multimedia (YouTube, camera), gaming (Angrybirds, Temple Run), social media (Hangouts, Whattsapp), to browsing (Chrome). Many of these subsystems are power hungry and drain a lot of battery energy in executing the apps causing inconvenience to users in forcing them to charge their batteries more than once a day. Apart from limited battery charge time, tight form factors of the phone coupled with the absence of any active cooling mechanisms (such as fans) makes the thermal management crucial [1] to (i) maintain the circuit integrity, (ii) avoid inconvenience to user.

CPU, GPU and display [2], [3] are still the primary contributors to both power consumption and temperature. Indeed, CPU and GPU are the device core computational elements, thus they have high switching activities. When operated at high frequencies and voltages, they exacerbate higher dynamic power consumption. Moreover, it is well known that temperature depends on on-chip power density [4]. Display has a significant contribution to power consumption due to its relative large dimension.

The focus of mobiles designers and developers is shifting from high performance to high user experience [5]. The quality of user experience is a broad concept which depends on a large number of variables, from personal tastes and preferences, to device operating conditions, to the applications currently running. Even though user experience is hard to measure, we can easily define it as the situation in which device behavior meets user expectations. Given this definition, it is easy to notice that expectation, thus experience, changes depending on which application is executing in foreground on the mobile device. For example, the level of expectation of a user playing games is much higher than that of a user browsing through websites. As a consequence, the level of performance that the device should provide to meet user expectation is different in the above two scenarios (higher for gaming, lower for browsing). This means that in this example we can obtain a power reduction by simply operating the device at a lower performance when browsing and at a standard performance when gaming, without impacting user experience. Potential for power savings also comes from change blindness. Change blindness is defined as the inability of users to notice small changes in brightness of displays over large time intervals. Therefore, even if starting from a standard brightness at the beginning of a session, brightness can be slowly reduced to save power [2], [6].

Power management techniques which have been recently proposed in literature do address user experience, but they have few major limitations that make them impractical, e.g. (i) the techniques are very specific to certain applications [7], [8]; (ii) some are not feasible to implement or they are not compatible with the existing standard approaches, thus they would require expensive OS restructuring [9], [10]. For such reasons, the actual power management of mobile devices has not changed much over the years.

Thermal management is a very actively researched area, spanning from high performance multi-clusters to mobile devices [1], [11], [12]. Thermal management can be broadly classified into reactive, if decisions are based on current or past temperature, or proactive, if decisions are based on predicted future temperature. Proactive thermal management requires a thermal model for predicting future temperatures. Most of the thermal modeling techniques proposed in the literature, however, use power measurements to predict temperature. Unfortunately, in mobile devices such power measurements are not available, as they do not have power sensors. An alternative involves deriving power through power models, which depend on the device performance metrics. However, the power models

MobiCASE 2014, November 06-07, Austin, United States

DOI 10.4108/icst.mobicase.2014.257788

suffer from inaccuracies and could be too computationally expensive for runtime management.

Considering this background, we can identify some limitations in the standard power and thermal management schemes present in mobile devices today. Assuming the Android OS as a reference, power management at the Linux kernel level is actuated by frequency governors, which switch voltages and frequencies at a high rate to match workload requirements and meet the set goals, such as high performance or low power consumption. Today, the standard governor, called Ondemand, scales frequency according to CPU utilization [13]. On the other hand, thermal management is handled at the Userspace level. In Qualcomm-based devices such as the Google Nexus 5, temperature is controlled by a daemon called Thermal Engine (TE), interfaced with temperature sensors. If a certain temperature threshold is exceeded, the TE reacts by applying a set of actions defined in its configuration file, such as limiting the frequency range or shutting down the device. This power and thermal management infrastructure has the following limitations:

1) Power and thermal management are not coordinated.
2) Thermal management is purely reactive.
3) Power management is agnostic of which applications are currently running and treats all of them in the same way.
4) Power management does not account for user-experience.

In this work we propose a novel user-centric proactive joint power and thermal management, designed for mobile devices, with the following contributions:

1) We formulate the management problem to account for user-experience.
2) We design management to be proactive and implement temperature prediction on the device itself.
3) We develop a general procedure to derive a simple yet accurate thermal model of the entire phone, based on observable quantities like CPU and GPU frequencies, and display brightness. Thus, it does not require actual power measurements. Our thermal model has a mean error of 1.17°C.
4) We exploit change blindness in our comprehensive power and thermal management.
5) We propose a lightweight and ready-to-use implementation for the manager, which is compatible with the standard Android environment and requires no operating system (OS) modifications.

We test our management techniques in comparison with the standard available power and thermal management schemes on a real Android device, the Google Nexus 5, with a set of most commonly used applications. The results shows that our solution can achieve up to 46% of application-specific power saving and up to 35% of average power saving at the device level.

The remaining of this paper is organized as follows: Section II presents the related work, Section III discusses the thermal model, Section IV describes the management formulation and implementation, Section V presents the experimental results and Section VI concludes this paper.

II. Related Work

The attention of both industry and research on user experience in addition to performance has grown consistently in the last decade, and many publications aim at achieving both high user experience and energy efficiency for mobiles. Various papers aim at measuring and modeling user experience [2], [5], [14] by collecting user feedback and correlating with users' personal profile. The message that emerges from this area of research is twofold: (i) it is extremely challenging to measure and model user experience and (ii) since the primary focus in mobile phones is to ensure quality user experience, power management should account for it.

For this reason some proposed techniques adopt a practical strategy and allow users to configure power management based on personal preferences. The authors of [10] proposes a novel task allocation infrastructure, which accounts for user experience as the user can assign a level of priority to applications. Similarly, authors of [15] propose a way to identify and select different priority levels for applications and suggest how this can be used to optimize power management.

Another class of approaches, instead of providing configuration interface to the user, model user experience depending on perceived delays and alternation of activity and idle periods. Reference [16] proposes an event-driven power management framework which increases CPU frequency in response to interactive events, in order to minimize users perceived delay. Work in [17] presents a power management strategy which uses hardware timers to capture idle and activity time intervals. A model for typical user session activity is proposed in Reference [2]. The authors then use the proposed model to compare various power management strategies. Also, the above work exploits change blindness to reduce display power consumption. The technique described in [18] employs a novel scheduling algorithm called energy-based fair queuing (EFQ), which achieves energy proportionality in battery-constrained systems while improving user experience. Another power efficient scheduling approach is presented in Reference [9]. The approach takes advantage of heterogeneous platforms to allocate tasks with different energy impact. While they achieve better energy efficiency, these approaches require an event-detection and timing infrastructure which involves modifications at the OS level. This makes the implementation challenging and can affect the portability of the framework between different devices. This is even exacerbated when the power management policy involves task allocation. In fact, the scheduler is one of the most critical sections in an OS. Modifying it is risky and time expensive [19].

Some recent publications address power management in mobile phones for specific applications. Reference [8] presents a unified dynamic voltage scaling (DVS) algorithm for CPU and GPU, which takes advantage of the execution profiles of 3D games to improve energy efficiency. Power management techniques presented in [20], [21] also reduces power consumption for gaming by adapting to current and predicted program state. Work in [22] optimizes power consumption for YouTube by intelligently scheduling download activities. The technique in [7] adopts tone mapping technique to adapt brightness and reduce LCD backlight level for mobile games, without compromising user experience. Even though these techniques successfully target power reduction and user expe-

rience, they are developed specifically for a target application, thus they cannot be generalized.

Thermal management is a broad and well investigated area of research, especially in the field of high performance multiprocessors [1], [12]. However, few publications target thermal management for mobile devices. In Reference [11] the authors presents a thermal model of a mobile system and use it to develop a thermal management strategy. The thermal model presented in [23] focuses on the thermal interaction with the battery subsystem. None of the above works present a coordinated power and temperature management solutions. To the best of our knowledge, our proposed work is the first protype of a ready-to-use implementation of a user-aware proactive joint power and thermal management technique.

III. Thermal Modeling Methodology

Most smartphones consists of heterogeneous computing elements equipped with many temperature sensors. Such smartphones have a set of operating conditions $\mathbf{F}$ that can be controlled at runtime and have significant influence on device power consumption and temperatures at various locations of the phone. Examples of these operating conditions are CPU core frequencies, GPU frequency, and display brightness.

For this paper, we control only the CPU frequencies, GPU frequency, and display brightness as CPU cores, GPU and display are the major contributers to the power consumption, devices temperature and user-experience [2]. The above set of operating conditions are denoted by $\mathbf{F} = [f_{c0}, f_{c1}, f_{c2}, f_{c3}, f_{GPU}, \beta]$, where f_{ci} is the frequency of CPU core i, f_{GPU} is the GPU frequency, and $\beta \in \{0, 1, \ldots, 255\}$ is the display brightness. Indeed, most smartphones today have embedded multi-core processors with per-core voltage and frequency control capability [24]. Note that all vectors and matrices are denoted in **bold**.

The reference device used in this paper is Google Nexus 5. This device has a quad-core Qualcomm's Krait processor with Adreno 330 GPU. The smartphone has 11 temperature sensors, referred to as *thermal zones*, whose values are exposed through `sysfs` interface of Linux kernel. The set of temperatures are denoted by vector $\mathbf{T} = [T_0, \ldots, T_{10}]$.

The thermal modeling for a processor is well described by the compact thermal modeling techniques such as HotSpot [25], which uses electro-thermal analogy to describe heat spreading and storing phenomena. The simplified state space representation [1], [12] is shown below:

$$\frac{d\mathbf{T}(t)}{dt} = \mathbf{A}'\mathbf{T}(t) + \mathbf{B}'\mathbf{P}(t), \tag{1}$$

where $\mathbf{A}'$ and $\mathbf{B}'$ are constant matrices. They are related to the thermal conductance ($\mathbf{G}$) and capacitance ($\mathbf{C}$) matrices as follows: $\mathbf{B}' = \mathbf{C}^{-1}$; $\mathbf{A}' = -\mathbf{B}'\mathbf{G}$. Power consumption is represented by $\mathbf{P}$. Note that it is not necessary to have same dimensions for both $\mathbf{T}$ and $\mathbf{P}$. Discretizing the above equation for a constant sampling interval of Δt, and we get

$$\frac{\mathbf{T}_k - \mathbf{T}_{k-1}}{\Delta t} = \mathbf{A}'\mathbf{T}_{k-1} + \mathbf{B}'\mathbf{P}_k, \tag{2}$$

$$\mathbf{T}_k = ((\mathbf{A}' - \mathcal{I})\mathbf{T}_{k-1} + \mathbf{B}'\mathbf{P}_k)\Delta t. \tag{3}$$

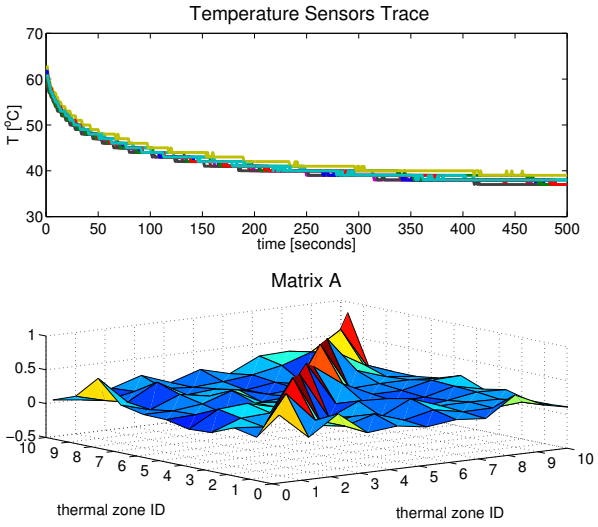

Fig. 1. Top figure: Cooling transient temperature trace; Bottom figure: matrix $\mathbf{A}$

Here k indicates the k^{th} time instant, and $\mathcal{I}$ denotes identity matrix of the same size as $\mathbf{A}'$.

Assuming that the operating conditions have linear dependency with power without significant loss in accuracy, since many commercial mobile devices do not have power sensors, i.e. $\mathbf{B}\mathbf{F}_k = \mathbf{B}'\mathbf{P}_k\Delta t$, and replacing $\mathbf{A} = (\mathbf{A}' - \mathcal{I})\Delta t$ results in the following alternative thermal model equation:

$$\mathbf{T}_k = \mathbf{A}\mathbf{T}_{k-1} + \mathbf{B}\mathbf{F}_k. \tag{4}$$

The advantage with the above modeling approach is that the operating conditions are usually exposed to the application layer and therefore the operating conditions can be modified without the need for any operating system (OS) modification.

To extract the above thermal model for a real device, such as Google Nexus 5, a simple profiling tool was developed which periodically samples temperatures and operating conditions. The sampling rate was set to 1 second. This is implemented as a bash script that runs on the phone and can be launched with a terminal emulator or an application manager. The tool reads the files related to the 11 temperature sensors and those related to the 6 selected operating conditions mentioned before, and then it writes values to a file. The output file is then fed to a least-square solver implemented in MATLAB to derive matrices $\mathbf{A}$ and $\mathbf{B}$.

Extracting the model essentially consists in computing matrices $\mathbf{A}$ and $\mathbf{B}$. The procedure we propose involves solving two least squares regression problems similar to [12]. The first problem is obtained by imposing $\mathbf{F} = \mathbf{0}$, which gives:

$$\mathbf{T}_k = \mathbf{A}\mathbf{T}_{k-1} \tag{5}$$

The solution of the above equation gives matrix A. This can then be substituted in (4) and solved for $\mathbf{B}$. The computing of matrices $\mathbf{A}$ and $\mathbf{B}$ is only once for a given phone, as the coefficients are dependent only on the geometry of the phone and the material composition. Also note that the computation

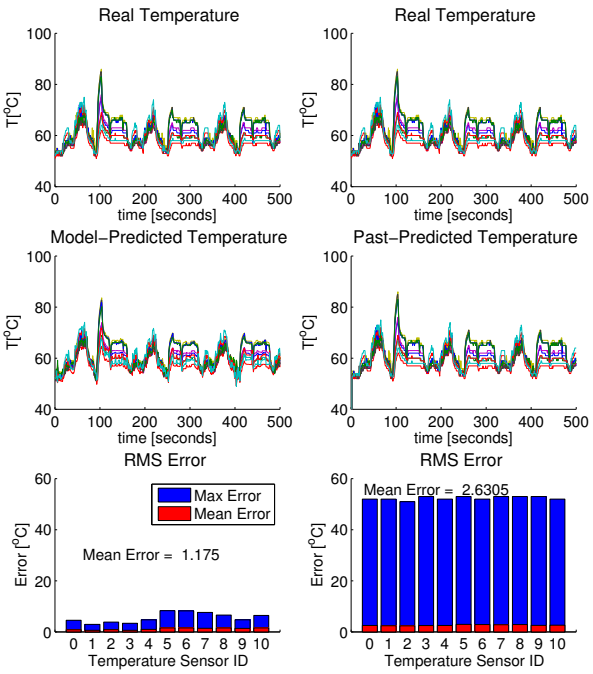

Fig. 2. Results showing the accuracy of the proposed thermal model

is done offline as the resources available on a mobile device are limited.

Figure 1 shows a sample temperature trace used for determining matrix $\mathbf{A}$ (top figure), and the resultant matrix is shown in the bottom figure. Note that in practice it is not possible to set $\mathbf{F} = \mathbf{0}$, as this would mean switching off the device. To obviate this, we set $\mathbf{F} = \mathbf{F}^{min}$, where $\mathbf{F}^{min}$ is the set of minimum operating conditions that guarantee functionality for the profiling tool, e.g., $\mathbf{F}^{min} = [f_{c0}^{min}, 0, 0, 0, 0, 0]$. As shown in Figure 1, the temperature trace used to derive $\mathbf{A}$ should capture the device cooling rate. On the other hand, the trace recorded for deriving $\mathbf{B}$ should capture the real user activity. Results for matrix $\mathbf{B}$ are analogous to that shown in Figure 1.

Figure 2 shows the case of a trace used to validate the model and its accuracy, which is expressed in terms of root mean square error (RMSE). The trace consists of temperatures collected for 500 seconds of execution of AnTuTu benchmark for 3 consecutive runs [26]. The top two plots show the real temperature trace, while the middle two plots show the model-predicted temperature and the predicted temperatures using the immediate previous temperature, respectively.

The model-predicted temperature is derived thanks to the thermal model described in Equation (4). Most reactive thermal managmenet schemes predict temperatures using the immediate previous temperatures.

Finally, the bottom two plots show the maximum and the mean prediction errors obtained in the above two cases. From the plots, we can see that our model-based prediction has a mean RMSE error of **1.17**°C, which is an improvement of **55**% w.r.t. previous temperature-based prediction. Also, our model achieves **80**% reduction in peak error w.r.t. previous temperature-based prediction.

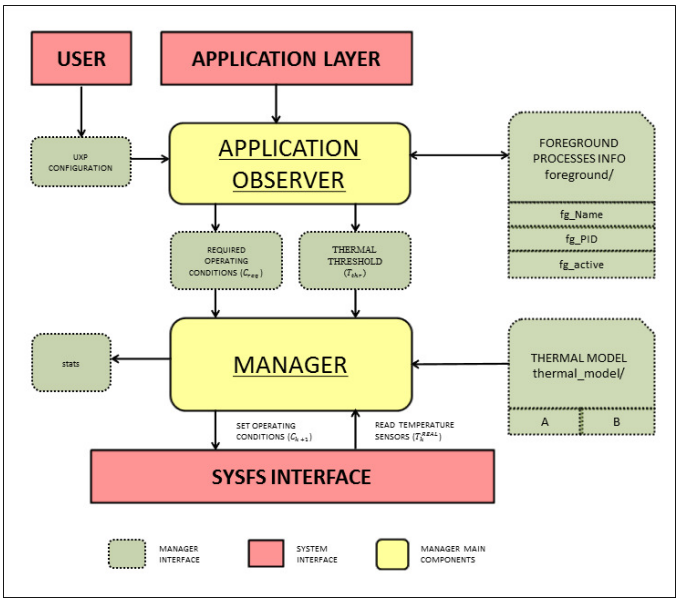

Fig. 3. Block diagram of the proposed power-thermal manager

IV. Management Formulation and Implementation

In this section, we present a formulation of joint power-thermal management problem, propose a novel solution, and describe the power-thermal manager which implements the solution. The joint power and thermal management problem formulation is described below:

$$\min_{\mathbf{F}_{k+1}} \quad \mathbf{F}_{k+1}^{req} - \mathbf{F}_{k+1}, \tag{6}$$

$$s.t. \quad \mathbf{T}_{k+1} \leq T_{thr}. \tag{7}$$

In the above formulation, $\mathbf{F}_{k+1}$ represents the set of operating conditions that will be applied at the next time step and it is the output of the management problem. $\mathbf{F}_{k+1}^{req}$ is the set of required operating conditions at the next time step, and it is a function of the application executing in foreground and of the level of user expectation. We assume that if $\mathbf{F}_{k+1} \geq \mathbf{F}_{k+1}^{req}$, then the user experience is met. The term $\mathbf{T}_{k+1}$ represents the vector of system temperatures at the next time step, which are predicted using the model presented in Section III. The temperature threshold T_{thr} is also a function of the foreground application and of the user expectation. In this work we consider the threshold to be a single value, but we can extend the idea by defining one threshold per each sensor. Note that a different choice of vectors $\mathbf{F}$ and $\mathbf{T}$ is consistent with the problem general formulation. Finally, note that the main goal of such a management is never to give more performance than what is required from a users perspective. This is the key source of power reduction in our solution.

Figure 3 shows the block diagram of the proposed management technique. The main components are the Manager and the Application Observer. The proposed implementation runs in Linux userspace and requires no modifications at the operating system level. In the following, we describe the main components in further details.

A. Sysfs Interface

The Sysfs Interface is the main interface between Linux kernel and the userspace, in which the OS kernel exposes data such as current operating conditions and temperature sensors values. In current versions of Android/Linux it is also possible to set operating conditions through the sysfs interface, such as CPU frequency, GPU frequency and display brightness. Therefore, the power and thermal management can be completely implemented in userspace, without the need of modifying the OS.

B. Manager

The Manager is the component devoted to finding the optimal operating conditions for the mobile device. When starting execution, it loads matrices $\mathbf{A}$ and $\mathbf{B}$. The two matrices are computed offline according to the procedure described in Section III.

Then the manager wakes up at a fixed rate and reads current temperatures $\mathbf{T}_k$ from the sysfs interface. It also reads the required operating conditions $\mathbf{F}_k^{req}$ and the thermal threshold T_{thr} from the user experience configuration files. Using these data it first computes $\mathbf{T}_{k+1}$ using Equation (4), then it computes the operating conditions $\mathbf{F}_{k+1}$ by solving the problem described in the Formulation (6)–(7) and applies them by writing values to the sysfs interface. The manager also writes data to the `stats` file, for debugging purposes. The Manager source code is written in C and it is cross-compiled with Android Native Development Kit (NDK) tools to run on our reference device, the Nexus 5 [27]. The default activation rate is 1 second, as this is the minimum rate that can be achieved in the userspace. To make the manager as less intrusive as possible, we decided to set a maximum bound on CPU and GPU frequencies, rather than fixing a constant value. This design choice guarantees the compatibility with the kernel-level power management schemes actuated by frequency governors. Indeed, these modules activate at a much higher rate (the Ondemand governor activates every 20 ms) [13]. The solution to the management problem to determine $\mathbf{F}_{k+1}$ through a simple heuristic algorithm. The manager first predicts temperature by computing Equation (4) with $\mathbf{F}_{k+1} = \mathbf{F}_{k+1}^{req}$. If the required operating conditions $\mathbf{F}_{k+1}^{req}$ are such that they do not cause temperature to exceed the threshold, then the manager applies $\mathbf{F}_{k+1} = \mathbf{F}_{k+1}^{req}$. Otherwise $\mathbf{F}_{k+1}$ is reduced and predicted temperature is checked again until the threshold is met. The strategy to reduce $\mathbf{F}_{k+1}$ and take advantage of the fact that CPUs in mobile devices have predefined voltage and frequency operating points. GPU frequency is not reduced, because the manager assumes that if the required GPU frequency is high, then it means that the foreground application involves graphics and requires the requested performance. Therefore penalizing the GPU may be unpleasant for the user.

Display brightness is handled separately, according to the concept of change blindness. When a new application is launched, display brightness β is set to the required value. After that, the manager progressively reduces it until reaching a predefined minimum value β^{min}. The rate at which β is reduced and the minimum value β^{min} are parameters that can be tuned in the manager configuration. The chosen values should guarantee that the user does not perceive brightness changes and degrade user experience [2].

C. Application Observer

The Application Observer essentially monitors which application is currently running in the foreground on the device. Based on this and on the User Experience (UXP) configuration provided by the user, it periodically updates the required operating conditions $\mathbf{F}_{k+1}^{req}$ and the thermal threshold T_{thr} for the current execution, by writing them to the appropriate file locations. The source code for the application observer is also written in C and cross-compiled with NDK tools. In the current implementation, the Application Observer wakes up at a fixed rate (default is 1 second) and executes the `top` command. The output of this command is the list of running processes with their properties. In particular the PCY field in the top denotes whether a process is in foreground (flag "fg") or in the background (flag "bg"). Usually there is not only a single process with an fg flag, but rather there is a list of them, and the application executing on the display is one of them. For this reason the observer parses the output of top and writes the names of all foreground processes to the file fg_Name, and the list of process IDs to the file fg_PID. Then it compares the list of current foreground processes with the UXP configuration. The UXP configuration contains the list of required operating conditions configured by the user for a set of applications, together with the desired thermal threshold. If any application in the UXP configuration appears in the fg_Name file, then the corresponding required operating conditions are written to the file containing $\mathbf{F}^{req}$, and the corresponding thermal threshold is written in the file containing T_{thr}.

D. The User Experience (UXP) Configuration

The UXP configuration is obtained by the configuration manager which requires a user to execute an application of his/her choice under the maximum operating conditions. While the user is executing the application in foreground, the configuration manager progressively lowers the operating conditions. When the user notices a degradation in the user experience of the application, s/he notifies the configuration manager, which registers the least acceptable operating conditions for that application. Alternatively, the procedure can start from the minimum operating conditions and increase them until the user experience of the application behavior reaches acceptable levels. The user is also asked to choose the default operating conditions for the device. These are used in the case the Application Observer does not find any correspondence between the list of foreground processes and the applications in the UXP configuration. The approach though it requires the user's effort (relatively low) in setting up the UXP configuration, it has the advantage of being personalized for that user. Such a provision is not available in other UXP model-based approaches such as [16], [17].

Finally, thermal thresholds are selected based on the accepted values typically used for thermal management, e.g. Qualcomm's Thermal Engine daemon.

E. Manager Installation and Execution

To install the management framework on an Android device it is just required to copy the executables for the Manager

and for the Application Observer to a location in the phone file system. After that it is sufficient to launch both and let them run in the background. To insure the correct behavior, other power or thermal management program running in the Userspace should be stopped first. However, we have verified that the manager is compatible co-running with Qualcomm's mpdecision. This is a service that automatically handles CPU hotplug based on computational load.

V. Results

The target platform for the evaluation of the proposed solution is a Google Nexus 5 smartphone. This device is equipped with the Qualcomm Snapdragon 800 chipset, featuring a quad-core Krait 400 CPU. The four cores have a range of frequency from 300 MHz up to 2.26 GHz, with predefined voltage and frequency operating points. It also has an Adreno 330 GPU, with frequency from 200 MHz up to 450 MHz. The display is a Full HD with In-Plane Switching (IPS) LCD [28] with 255 brightness levels. The phone was rooted to get root privileges to control the frequencies of CPU and GPU, and display brightness.

The evaluation has been conducted selecting applications among the most popular ones: Chrome, Gmail, Angrybirds, Hangouts, Dialer (standard phone call), Camera, YouTube [14], [29].

To ensure that comparisons are consistent and reproducible, we wrote an application that allows to record and replay sequences of events on the device (e.g. touches on the display, pressing the power button, pressing the volume button, etc.).

In this device they are already executing by default two other Userspace managers: the Thermal Engine and mpdecision. Moreover, the standard Ondemand governor is operating at the Kernel level. The Thermal Engine also operates by limiting the maximum operating conditions, based on current temperature. Therefore, to avoid interference with our manager, it should be stopped. Unfortunately, this program cannot be stopped, therefore it is necessary to reduce its functionalities by replacing its configuration file with an empty file. As for mpdecision, this program essentially switches on and off CPU cores based on utilization metrics. Since it does not perform frequency scaling, it should be kept active and compatibility should be guaranteed.

Figure 4 reports an extract of 20 seconds of an execution trace monitored while applying the proposed manager, to describe the details of working of the propsed manager. For this experiment we are executing Google Chreme application with the required operating conditions $\mathbf{F}^{req}$ = [1 GHz, 1 GHz, 1 GHz, 1 GHz, 200 MHz, 150]. The first plot reports the temperature traces of the 11 sensors present in our device. For simplicity, no legend is reported here. Note also that in general the exact location of sensors is not disclosed. In this case, the thermal threshold was set to 80°C, therefore temperature is kept in a safe range. The second plot reports the behavior of CPU and GPU frequencies. Once the application is launched (at 2 sec), the Application Observer detects it running in foreground and selects the corresponding required operating conditions. No thermal management is activated in this case, therefore the manager always selects $\mathbf{F}_{k+1} = \mathbf{F}^{req}_{k+1}$. The black dotted line in the second plot represent the required

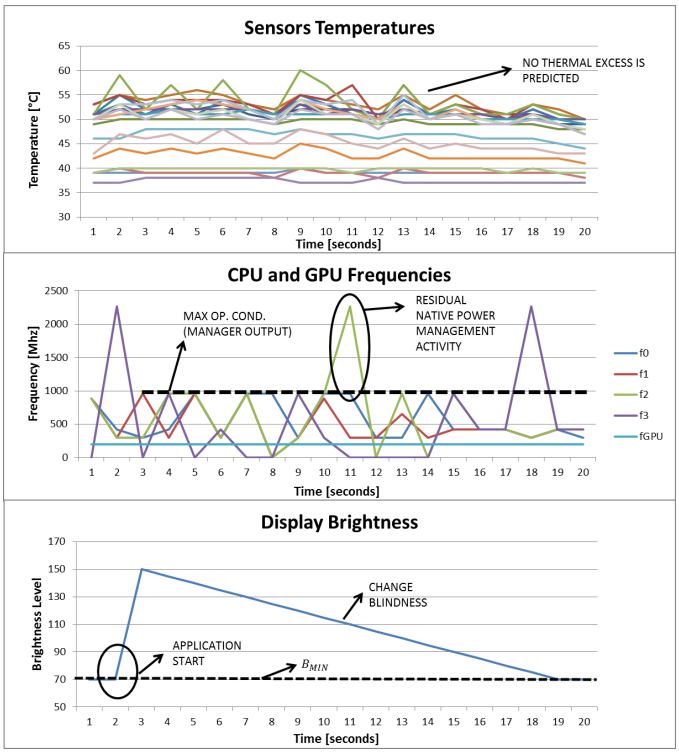

Fig. 4. Sample execution trace of temperature and CPU-GPU frequencies under the deployment of the proposed manager

CPU operating frequency, which is the maximum bound set by the manager. Indeed, the Ondemand governor at the Kernel level is free to switch frequency at a higher rate. Moreover, when a CPU core frequency is equal to zero, then it means that mpdecision has switched the core off. At 11^{th} second we can notice that the frequency of CPU 2 exceeds the bound imposed by the manager. This is due to the combination of Thermal Engine, mpdecision, and Ondemand governor activities. However, this does not affect the temperatures much. Finally, the third plot shows the behavior of display brightness. When an application is launched, the required level is set (150 in this case). After that, change blindness is exploited and brightness is progressively reduced by 5 every seconds until the predefined minimum value β^{min} is reached (70 in this case).

In Figure 5 we show what happens when the manager detects a thermal emergency (e.g. the threshold T_{thr} is exceeded by some sensor predicted temperature). While executing a sample trace of the popular game Angrybirds, we monitor the temperatures and the CPU maximum frequency (e.g. the output of the manager) in two different cases. In the first case (on the left) the thermal threshold for the target application is set at 70°C, in the second case, the threshold is 60°C. For simplicity, only the temperature of the two sensors which detect thermal excess is reported, namely T9 and T10. In both cases, the maximum operating conditions are modulated by the manager in order to keep the predicted temperature lower than the threshold. Note that the threshold can be tuned to provide different performance tradeoffs.

Figure 6 reports our results in terms of power savings. Table I shows the configurations used for various applications

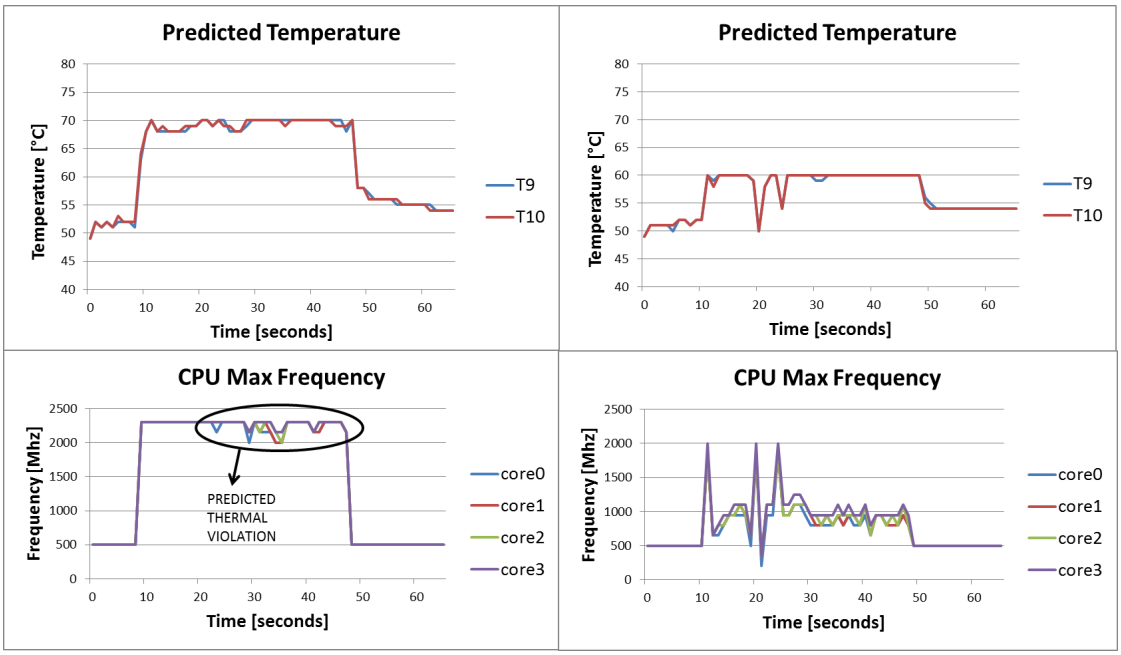

Fig. 5. Example of the proposed manager handling thermal emergenices

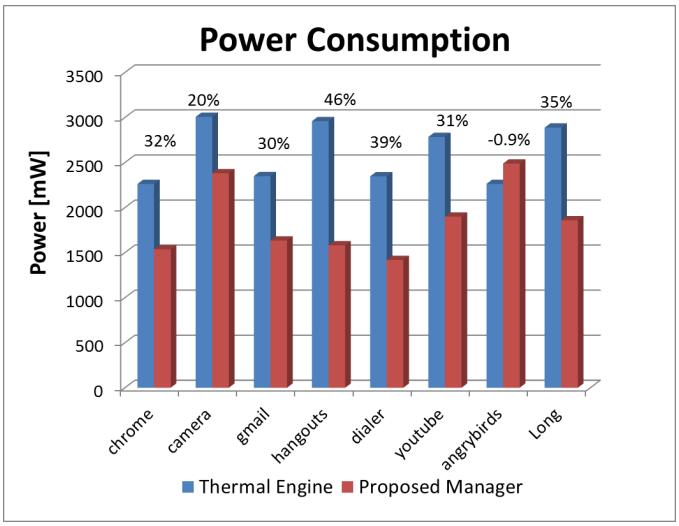

Fig. 6. Power savings resulting from the use of the proposed manager

in our evaluation of the proposed power-thermal manager. We record and replay event traces for each application and measure power consumption respectively with the default Thermal Engine and with the proposed user-centric manager. In both cases, mpdecision is active and the Ondemand governor is set. Power consumption is monitored by sampling the battery voltage and current values at the sysfs interface. In devices where this is not available, it is required to use an external power monitor. The traces are for a single application with 1 minute duration. Then we also compare power consumption on longer traces which combines all the considered applications (labeled as "Long" in the bar graph), which have 10 minutes duration. The values of power consumption in Figure 6 are average values.

TABLE I. CONFIGURATION VALUES USED FOR VARIOUS APPLICATIONS IN THE PROPOSED MANAGER

Application	CPU freq. (MHz)	GPU freq. (MHz)	Display brightness
Chrome	1000	200	150
Camera	1500	350	150
Gmail	800	200	70
Hangouts	900	200	100
Dialer	1300	200	55
YouTube	1500	450	150
Angry Birds	2260	450	255
Default	500	200	100

For all the considered applications, our manager provides a power saving w.r.t. the standard power management, up to 46% in the case of Hangouts. The only exception is the case of Angrybirds. Indeed, in this case the user expectation is very high, therefore the required operating conditions are set at the maximum. Finally, in the case of Long traces, our manager provides up to 35% average power saving w.r.t. the standard power management.

VI. CONCLUSION

In this paper we proposed and presented a ready-to-use solution for user-centric proactive power and thermal management. The proposed approach is able to adapt to maximize user experience and reduce power consumption while keeping temperature in a safe range. Temperature prediction is based on a novel observable-based thermal model which achieves a mean accuracy of 1.17°C.

The proposed technique has been implemented on a real Android device and tested on a set of real life applications. The experimental results show that our manager achieves up to 46% reduction in application-specific power consumption and up to 35% reduction in average power consumption, when compared to standard power and thermal management.

REFERENCES

[1] K. Sekar, "Power and thermal challenges in mobile devices," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 363–368.

[2] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 168–178.

[3] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21.

[4] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 170–183, Jan 2013.

[5] B. Chihani, K. ur Rehman Laghari, E. Bertin, D. Collange, N. Crespi, and T. Falk, "User-centric quality of experience measurement," in *Mobile Computing, Applications, and Services*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, G. Memmi and U. Blanke, Eds. Springer International Publishing, 2014, vol. 130, pp. 33–46.

[6] P. S. Guterman, K. Fukuda, L. M. Wilcox, and R. S. Allison, "75.3: Is brighter always better? the effects of display and ambient luminance on preferences for digital signage," *SID Symposium Digest of Technical Papers*, vol. 41, no. 1, pp. 1116–1119, 2010.

[7] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan, "Adaptive display power management for mobile games," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 57–70.

[8] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated cpu-gpu power management for 3d mobile games," in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, ser. DAC '14. New York, NY, USA: ACM, 2014, pp. 40:1–40:6.

[9] V. Petrucci, O. Loques, and D. Mosse, "Lucky scheduling for energy-efficient heterogeneous multi-core systems," in *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*. Berkeley, CA: USENIX, 2012.

[10] N. K. Nithi and A. J. de Lind van Wijngaarden, "Smart power management for mobile handsets," *Bell Lab. Tech. J.*, vol. 15, no. 4, pp. 149–168, Mar. 2011.

[11] F. Paterna, J. Zanotelli, and T. S. Rosing, "Ambient variation-tolerant and inter components aware thermal management for mobile system on chips," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 210:1–210:6.

[12] V. Hanumaiah, D. Desai, B. Gaudette, C.-J. Wu, and S. Vrudhula, "Steam: A smart temperature and energy aware multicore controller," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 5s, pp. 151:1–151:25, Oct. 2014.

[13] V. Pallipadi and A. Starikovskiy, "The ondemand governor - past, present, and future," in *Proc. Linux Symposium*, vol. 2, 2006, pp. 223–238.

[14] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 179–194.

[15] M. Martins and R. Fonseca, "Application modes: A narrow interface for end-user power management in mobile devices," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:6.

[16] S. Kim, H. Kim, J. Hwang, J. Lee, and E. Seo, "An event-driven power management scheme for mobile consumer electronics," *Consumer Electronics, IEEE Transactions on*, vol. 59, no. 1, pp. 259–266, February 2013.

[17] G. Lim, C. Min, D. H. Kang, and Y. I. Eom, "User-aware power management for mobile devices," in *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, Oct 2013, pp. 151–152.

[18] J. Wei, E. Juarez, M. Garrido, and F. Pescador, "Maximizing the user experience with energy-based fair sharing in battery limited mobile systems," *Consumer Electronics, IEEE Transactions on*, vol. 59, no. 3, pp. 690–698, August 2013.

[19] P. Mercati, A. Bartolini, F. Paterna, T. Rosing, and L. Benini, "A linux-governor based dynamic reliability manager for android mobile devices," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–4.

[20] B. Dietrich and S. Chakraborty, "Power management using game state detection on android smartphones," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 493–494.

[21] ——, "Managing power for closed-source android os games by lightweight graphics instrumentation," in *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, Nov 2012, pp. 1–3.

[22] X. Li, M. Dong, Z. Ma, and F. C. Fernandes, "Greentube: Power optimization for mobile videostreaming via dynamic cache management," in *Proceedings of the 20th ACM International Conference on Multimedia*, ser. MM '12. New York, NY, USA: ACM, 2012, pp. 279–288.

[23] Q. Xie, J. Kim, Y. Wang, D. Shin, N. Chang, and M. Pedram, "Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 242–247.

[24] B. Calhoun and K. Craig, "Flexible on-chip power delivery for energy efficient heterogeneous systems," in *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, May 2013, pp. 1–6.

[25] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron, "An Improved Block-based Thermal Model in Hotspot 4.0 with Granularity Considerations," in *Proc. WDDD*, 2007.

[26] AnTuTu benchmark. http://www.antutu.com/en/Ranking.shtml.

[27] Android NDK. https://developer.android.com/tools/sdk/ndk/index.html.

[28] "Google nexus 5," http://en.wikipedia.org/wiki/Nexus_5.

[29] "List of most downloaded android applications," http://en.wikipedia.org/wiki/List_of_most_downloaded_Android_applications.