API for Power-Aware Application Design on Mobile Systems

N. Peters[1], S. Park[1], D. Clifford[2], S. Kyostila[2], R. McIlroy[2], B. Meurer[2], H. Payer[2], S. Chakraborty[1]
Technical University of Munich[1], Google Inc[2]

ABSTRACT

Thanks to the performance improvements in hardware and software architectures, more applications, which used to run on desktop computers, are now being migrated to mobile devices. However, this entails increased power consumption, that necessitates more effective runtime power management techniques due to battery capacity constraints. Such techniques should reduce power consumption while satisfying user-perceived requirements, such as frame rate, and response times. A major hurdle in incorporating such techniques into real products is that user-perceived requirements are only visible to user applications, but not accessible by the power managers residing in the operating system. In this paper, we show that better power management is achievable by passing such information to the OS, and propose an API for that purpose.

1 INTRODUCTION

One of the major decision criteria for buying a smartphone is its battery run time [5]. Hence, power management techniques for mobile devices have become very important in today's system design. Moreover, a key characteristic of applications running on mobile devices is that they are user experience-sensitive. For example, an application is expected to respond immediately to touch events. On the other hand, there is no need to rush executing work that is not perceivable by the user, which can increase the power consumption - such as background tasks. The major drawback of the current Android software architecture is that the power management is done in the kernel space without regards to application-specific characteristics. There have been prior works to reduce power consumption with minimal impact on the user experience for applications such as mobile web browsing [4, 6], gaming [1, 3], etc. Although they show the benefit of the interaction between the kernel and applications, these works are highly application-specific and require custom modifications to either kernel or user space applications.

In this paper, we show that there is a lack of support from the operating system (OS) to enable a user application to pass over user requirements or application-specific information to the kernel. Then, we provide two use cases, mobile gaming and web browsing, to motivate an API between the user space and the power managers in the kernel space for better CPU power management, as depicted in Figure 1. Such an API should be general enough to be applicable to a wide variety of applications. We believe that such a generalized framework allowing coordination of applications and the OS, would encourage incorporating previously proposed power reduction techniques to real products.

MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5712-8/18/05.
https://doi.org/10.1145/3197231.3197253

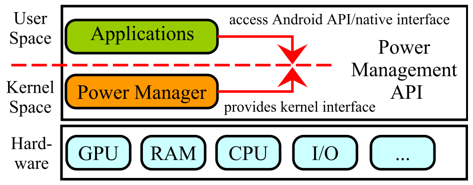

Figure 1: Possible Android system infrastructure for an application-aware kernel power manager interface.

2 API FOR POWER MANAGEMENT

In this section, we propose an API for power-aware application design in Android. We introduce two examples, namely mobile games and web browsing, where sharing information between application and governor results in substantial power savings and derive a generalized API based on these examples. Finally, we evaluate the consequences for developers when establishing such an API.

2.1 Application-Aware Power Management

Games show how target frame rate and timing deadlines can be exploited for interactive workloads. The browser work emphasizes the importance of proper workload prioritization, which can only be extracted from the application itself.

Game applications: Mobile games are very popular, but power hungry and computation intensive applications, because the complexity of modern games' graphics, physics and artificial intelligence (AI) is constantly increasing. The re-calculations have to be performed every game frame, what poses high workload on the system. The target frames-per-second (FPS) value is normally 30 or 60 to maintain a good user experience. From the gaming example, we identify two classes of information that can be passed from the applications to the governor, *frame rates* and *deadlines*. A simple approach to control the power consumption is to pass the target frame rate and the currently achieved frame rate to the governor. The power manager can regulate the CPU frequency based on the discrepancy of the achieved and the target frame rate. Further, the frame rate can be considered a deadline [1, 3]. The goal of the power management strategy is to calculate the frame (physics, etc.) within a particular time slot (e.g. 16.67 ms for 60 FPS). Predictive strategies can be applied to calculate the workload of the next frame based on the past workloads. This information can be used to find the appropriate frequency level for the particular frame that just fulfills the frame's resource requirements. Recent gaming works that implements such strategies report on average 41.9 % for modern mobile hardware architectures [3] compared to the Android default interactive governor.

Mobile web browsing: The web browser is another popular smartphone application [2] and we have shown that there is a large potential for power savings in mobile web browsing [4]. Browsing can be divided into different states such as loading a page, scrolling, etc. There is application-specific data (e.g. loading state) readily available within the browser. Based on that, we define information that can be passed from the application to the governor. For example, maximum response and loading times can be handled as *deadlines* if the prior workload can be determined. Otherwise, both actions can be treated as high *priority* actions that need to be completed as fast as possible. The priority of the current action can be used to distinguish between high workloads that are critical (foreground load) and others that are not necessarily relevant for a good user experience (background load). Such information can solely be provided by the application and by no other entity in the system. Further, the animation action requires a minimum FPS value to be maintained. Scrolling or zooming actions, but also videos playback are considered animations. Similar to gaming, the FPS value can be used as a *deadline* or as a target *frame rate*.

2.2 API between Applications and Kernel

As shown in the previous sections, there is a set of common information among Android applications that can be used for power management. Gaming and browsing already capture a wide range of application types. E.g. social media applications such as Facebook and Twitter can be compared to scrolling through a web page. Other applications, e.g. navigation, are graphics intensive such as games. In general, *all* user interactive applications need to maintain a *frame rate*, have high *priority* phases such as displaying incoming messages or other timing constraints such as *deadlines*. Hence, we suggest a power manager that can switch between different power management strategies, namely *modes*. The mode can be changed by the developer on demand. Besides one default mode (e.g. an Android default governor), the developer can choose the mode based on the state of the application and needs to provide mode-specific parameters as described in the following.

Frame rate: The examples in the previous section show that the FPS value can be exploited for power management for interactive applications or videos. As described, the frame rate can be seen as a target value and can be provided by any application that performs frame-based calculations. For example, there are games that run at 60 FPS, but nowadays there are also many games that target 30 FPS. Some applications might even target a lower frame rate. If the governor knew the target value, it could adjust the exact frequency level that is needed by the application. This is not possible by using workload information only. As of now, there is no possibility to acquire the current frame rate within the kernel without input from the application layer. The frame rate information within the kernel is obtainable from the GPU source code,that it is usually closed-source. Hence, the current FPS value has to be periodically passed from the application to the kernel to adjust the control loop.

Deadlines: Deadline information could be useful in combination with workload estimations. In many applications, there exists a temporal correlation of frame-based workload, which allows quite accurate workload estimations inside the governor [1, 3]. Generally, we can consider the vertical synchronization signal (VSync) of the display as a deadline for processing frames.

Workload Priority: From the browser power management work we have learned that workloads can be prioritized differently and, hence, require different power management techniques. For example, we can distinguish between foreground and background load for browsing, because the browser keeps track of the web page loading state. Generally, developers are aware of critical sections within their application. Hence, the prioritization of the workload can be a hint to the power manager. This would inform the power manager not to restrict any resources when the application is running. In the browser example, the foreground load can be marked as *high* priority while the background load can have a *low* or a *default* priority level. As a consequence, the power manager can tune its strategy as needed. The characteristics of the application workload are not obscure anymore, and the power manager can perform better power management.

2.3 Implementation Issues and Challenges

Providing a power-aware API between the application and the kernel layer requires modifications to the Android OS. However, numerous previous work point out that the potential power savings surely outweigh the overhead. One challenge is to integrate the API within the system. As shown in Figure 1, a *communication channel* between the application and the kernel needs to be established. Moreover, the hardware developers need to adopt their kernel software to provide this kind of API. This involves implementation of the power manager for individual hardware platforms. Finally, the application developers should have an idea of the actual resource demand to apply the API in the correct manner. Here, the automated retrieval of this information and the awareness of the developers should play a central role.

3 CONCLUDING REMARKS

Despite a considerable amount of prior work on power management for applications running on mobile devices, introduction of such techniques into real products has been sluggish. This is due to, on the one hand, lack of standardized means in Android systems to pass the application-specific information and user requirements to the operating system, and on the other hand, customization efforts required for implementing state-of-the-art power management techniques. In this paper, we observe two applications, games and web browsers, to identify information that are useful for power management done in the operating system, and propose an API for energy-efficient application development. As power consumption of mobile devices is of ubiquitous importance, efforts to push power saving techniques to the general awareness are s major concern.

Acknowledgments: This work was supported by Google Inc.

REFERENCES

[1] B. Dietrich and S. Chakraborty. Lightweight graphics instrumentation for game state-specific power management in Android. *Multimedia Systems*, 20(5), 2014.
[2] Google Inc. There's an app for that...the browser, 2015.
[3] N. Peters, D. Füß, S. Park, and S. Chakraborty. Frame-based and thread-based power management for mobile games on hmp platforms. In *IEEE International Conference on Computer Design (ICCD)*, 2016.
[4] N. Peters, S. Park, S. Chakraborty, B. Meurer, H. Payer, and D. Clifford. Web browser workload characterization for power management on hmp platforms. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.
[5] A. Pilon. Smartphone battery survey: Battery life considered important. https://aytm.com/blogmarket-pulse-research/smartphone-battery-survey/, 2016.
[6] Y. Zhu, M. Halpern, and V. Reddi. Event-based scheduling for energy-efficient qos (eqos) in mobile web applications. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015.