Evaluating Impact of Storage on Smartphone Energy Efficiency

David T. Nguyen
College of William and Mary
Williamsburg, VA 23185, USA
dnguyen@cs.wm.edu

Abstract
We present an experimental study of how storage techniques impact energy consumption in smartphones. We design and implement a system that tracks I/O activities of smartphones in real-time and dynamically changes storage configuration by matching I/O patterns in order to reduce energy consumption. Our system is evaluated on the 20 most popular applications from Android Market, and our results show that the optimal configurations save from 21% to 52% of battery life. We believe that they highlight a new and interesting direction in which the topic of smartphone energy consumption can be further evaluated and expanded upon.

Author Keywords
Dynamic storage configuration; I/O optimization; Smartphone energy-efficient system

ACM Classification Keywords
C.4 [Performance of Systems]: Design studies; C.5.3 [Computer System Implementation]: Microcomputers

General Terms
Design; Experimentation; Measurement; Performance

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.
ACM 978-1-4503-2215-7/13/09.

http://dx.doi.org/10.1145/2494091.2501083

Introduction

A common complaint among smartphone owners is the poor battery life. To many users, being required to charge the smartphone after a single day of moderate usage is unacceptable. This fact demonstrates the necessity for solutions which address the issue of energy consumption in smartphone devices. Therefore, we are motivated to investigate the direct impact of smartphone storage techniques on total battery consumption. The key research questions are: *How does storage affect smartphone energy efficiency?*, *What are the root reasons?*, and *How can we optimize smartphone storage in order to save more energy?*

In order to answer the research questions, we plan to evaluate smartphone power efficiency at various layers of the I/O path, and provide evidence which highlights that the energy consumption of a smartphone can differ depending on storage techniques employed. Different scheduling algorithms on the block layer or different queue lengths on the device driver may impact the total energy consumption differently. Our pilot solution finds for benchmarks the combinations of scheduling algorithms and queue lengths with optimal energy savings. The system prototype implemented on the Android platform tracks smartphone's I/O pattern in run-time, and matches with the benchmark with closest I/O pattern. After having matched with a benchmark, the system dynamically configures an optimal storage configuration to achieve lower energy consumption.

Thesis Statement

We summarize our thesis statement as follows:

- Investigate the impact of storage on smartphone energy efficiency.
- Explain root reasons of such impact.
- Develop storage-aware energy saving solutions.

Expected Contributions

If we succeed, we will contribute to a better understanding of the limitations of current day and future smartphone devices, i.e., how and why such devices exhibit significantly different energy efficiency when different storage policies are configured in the I/O path and what device and system improvements are necessary. We will also contribute specific innovative storage-aware energy saving solutions that work well in practice. New models and machine learning classifiers will be derived from actual measurement in a wide variety of setting and then used to remove or mitigate potential performance impact of storage-aware energy saving solutions.

Related Work

Kim et al. [6] present the analysis of storage performance on Android smartphones and external flash storage devices. Their discovery of a strong correlation between storage and application performance degradation serves as motivation for our work. Carroll et al. [5] measure the breakdown of energy consumption by the main hardware components in the device. Their direct measurements of each component's current and voltage are used to calculate power. This is done on a smartphone used for scientific purposes only, and many experiments cannot be replicated on commercially available smartphones. We take a different approach based on the precise analysis of the I/O activities between the application layer and the flash storage. Pathak et al. [8] introduce an application profiling approach in which they propose a system mapping energy activities to program entities based on estimates of routines' running time. The work, however, is done only on the application level.

Background and Motivation

In this section, we introduce background and motivation of our work. Next, we explain the measurement and

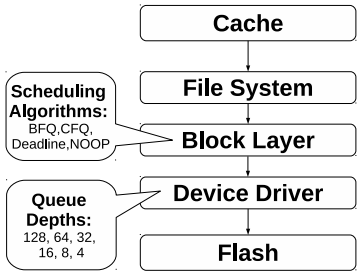

Figure 1: I/O Path.

methodology. Afterwards, we proceed with the measurement on the block layer and device driver.

I/O Path Components

In this work we take a look at energy efficiency of various storage techniques applied at several main components such as the block layer and device driver. In particular, our focus is to investigate the impact of different storage configurations on power level in smartphones. We illustrate the main kernel components affected by a block device operation on the I/O path in Figure 1. The figure is adapted from the literature [3]. At the block layer, the main work is scheduling I/O requests from above and sending them down to the device driver. The Linux kernels on Android smartphones offer 4 scheduling algorithms: BFQ, CFQ, Deadline, and Noop. In some Android phones, the default fixed scheduling algorithm is BFQ (e.g., Nexus One), others use CFQ (e.g., Galaxy Nexus, Nexus S). The device driver gets requests from the block layer, and processes them before sending back a notification to the block layer. On the device driver, we are interested in a parameter called *queue depth* that is defined as the number of pending I/O requests for storage. The queue depth is fixed to different values depending on vendors, usually 128 (e.g., Galaxy Nexus or Nexus One).

Benchmarks

We run 8 popular benchmarks on the Nexus One phone with Android platform under different storage configurations and measure power consumption levels with the Monsoon Power Monitor [2] (details given in Performance Evaluation). Each benchmark tests different phone subsystems and has its specific I/O pattern.

Block Layer

The default file system, scheduling algorithm, queue depth, and caching policy for the Nexus One is YAFFS2, BFQ, 128, and write-back, respectively. Each benchmark is executed on the phone for each scheduling algorithm and the power level is measured. The parameters are fixed to the default values, including the queue depth 128 and write-back caching policy. The results are illustrated in Figure 2. The first observation says that for the same benchmark, different scheduling algorithms result in different power levels. For instance, the AnTuTu (1st benchmark) average power consumption level is 792mW with CFQ, 720mW with Deadline, 792mW with Noop, and 1080mW with the default BFQ. Another observation is that none of the scheduling algorithms is optimal for all benchmarks. However, it is possible to find the optimal scheduling algorithm(s) for each benchmark and save relatively a lot of energy. For example, AnTuTu benchmark has the optimal power consumption with the Deadline algorithm, and more than 33% of energy on average can be saved compared to the default configuration with BFQ.

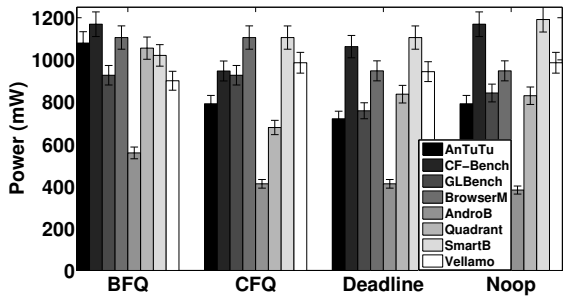

Figure 2: Power for Default Configurations.

Device Driver

To investigate impact of the device driver level on energy consumption, we run the benchmarks with different queue depths and compare how different queue depths affect

power levels. On Nexus One phone, the default queue depth is 128. The power consumption of this default queue depth is already illustrated in the previous Figure 2. Therefore, we investigate the power levels of the depth 4 in this section and compare with previous measurements. Figure 3 shows the power levels for the depth 4 normalized to the consumptions with depth 128. Looking at AnTuTu, with BFQ and queue depth 4 (BFQ/4) the average power consumption is 720mW which corresponds to 66.7% of the default BFQ/128 consumption. That means by changing the queue depth to 4, the phone can save on average 33.3% energy.

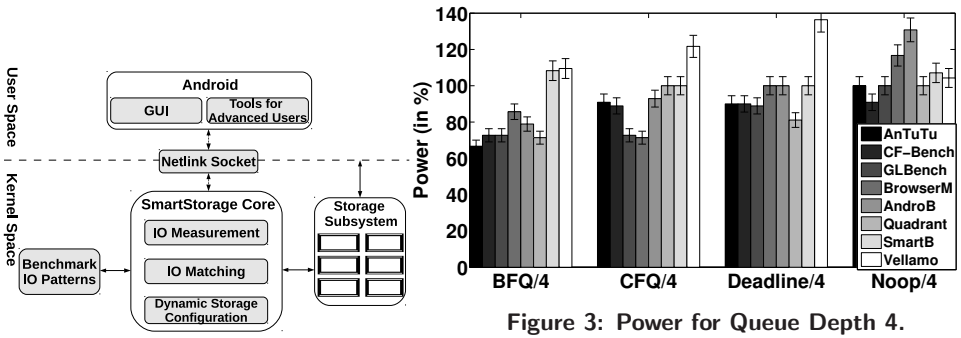

Figure 4: Architecture.

Figure 3: Power for Queue Depth 4.

Optimal Consumption

In order to find optimal power consumption for all benchmarks with above knowledge, we run for each benchmark all 8 possible combinations of scheduling algorithms (BFQ, CFQ, Deadline, Noop) with queue depths (128, 4) researched. This is recorded in a benchmark table.

Pilot Solution

In this section, we present a system prototype named SmartStorage [7]. From previous sections, for each benchmark there exists a combination of a scheduling algorithm and queue depth that is most power efficient. This information can be reused. First, we investigate the I/O pattern of each benchmark. Next, we obtain a run-time I/O pattern from the phone and match it to a benchmark with the most similar I/O pattern. Finally, an optimal combination of a scheduling algorithm and queue depth is configured. We discuss details in the following subsections.

The architecture in Figure 4 is divided into kernel space and user space. Kernel space consists of two main modules: SmartStorage Core and Benchmark I/O Patterns. User space includes the graphics user interface (GUI) and Tools for Advanced Users. Following, we elaborate detailed functionalities.

SmartStorage Core has three main functionalities. First, it obtains phone's run-time I/O pattern. Next, it gets a combination of a scheduling algorithm and queue depth with optimal power efficiency. Finally, it configures this combination in the block layer scheduler and the device driver of corresponding flash partitions.

The phone's run-time I/O pattern is obtained via blktrace [1, 4]. After gathering I/Os for a predefined time period, it calculates the run-time I/O pattern that is later used for matching with benchmark patterns. The pattern consists of rates of each I/O type per second. Note that such a pattern characterizes the I/Os of the whole phone, including those originating from background services. Therefore, this approach is not application-dependent.

Matching is done in the second phase after acquiring the phone's run-time I/O pattern. The phone's pattern is matched to a benchmark with the most similar I/O pattern. Since each benchmark has a combination of a

scheduling algorithm and queue depth with optimal consumption, that combination is returned as a result of this phase. With power efficiency in mind, we want a computationally inexpensive matching approach that is at the same time precise. Having all types of I/Os coming to storage, simple intuition says that what matters most at the end are the total number of completed reads and number of completed writes in a given interval. Furthermore, it is necessary to take into consideration differences between characteristics of read and write I/Os. Some partitions will serve reads better than writes or vice versa. Some partitions will be read-only, other allow both read and write. Motivated by this, we decide to expand the simple intuition, and do the matching based on the proportions of rates of completed reads and completed writes.

For clarity, let us define *RCRate* as *number of reads completed per second.* and *WCRate* as *number of writes completed per second.* Further, let us define *Rate Proportion (RP)* as *RP = RCRate / WCRate*. If the Rate Proportion (RP) of the phone's I/O pattern is close to the RP of a benchmark, a match is found. Finally, the optimal scheduling algorithm is set in the block layer scheduler and the optimal queue depth is set in the device driver.

Performance Evaluation

This section evaluates the SmartStorage efficiency by comparing energy usage of the 20 most popular applications from the Android Market with and without SmartStorage.

In our experiments, we use the SmartStorage implementation in the Nexus One phone. To measure energy consumption, the Monsoon Power Monitor [2] is utilized. We run the experiments with the top 20 free applications from the Android Market as of August 7, 2012. During our experiments, all radio communication is disabled except for WiFi. The screen is set to stay awake mode with constant brightness and auto-rotate screen off. When SmartStorage is in use, it runs only in the background and its GUI is off.

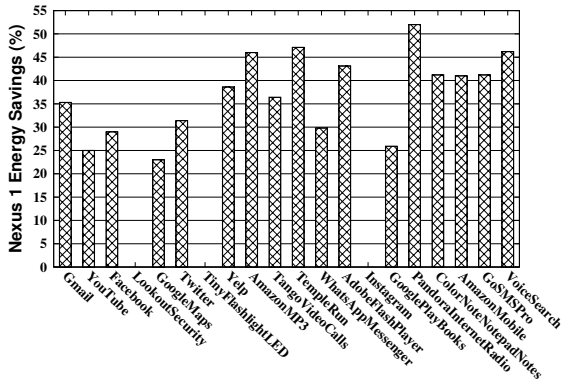

Figure 5: Energy Savings on Nexus One.

In order to address how much energy our solution saves in a typical use case, we run each of the 20 applications mentioned with SmartStorage in the background and compare with the case when the application is running with the default scheduling algorithm and queue depth (BFQ/128). A typical use case varies for applications. For instance, for Gmail, we read 20 emails and write 10 emails; for Amazon Mobile, we search for 20 products and read information about them; in Pandora, we listen to a channel for 30 minutes; on YouTube we search and listen to 5 songs; on Facebook, we read and write posts, etc. The Android Monkey tool is utilized to allow repeating the same behavior more times with and without SmartStorage so as to ensure fairness. The results of the total savings are in Figure 5. We can observe that the energy savings

vary from 23% to 52% and the largest savings are with Pandora application (52%). The three applications with no energy values have the optimal configuration identical to the default parameters of the phone.

Remaining Steps

In the device driver layer, we benchmarked the phone with two different queue depths and found significant differences in energy consumption. Naturally, more research on combining queue depths and scheduling algorithms may yield higher savings. We proposed the *RP* metric that proved to be efficient at matching I/O patterns. However, we plan to research a machine learning based method in the future. Our pilot solution periodically measures the storage I/O and then matches the I/O fingerprint to that of benchmarks for locating the optimal storage policy to save energy. If this process happens too frequently, the cost may be unnecessarily high and the system may not be stable since application performance may be impacted during highly frequent storage policy transitions, which we also plan to evaluate. If such a process happens too sparsely, we will not save much energy. Hence, we plan to monitor application events such as application started and terminated, and use them to adapt the measurement and matching frequency.

The conventional wisdom is that storage contributes little (approximately 30%) to the total power consumption [5]. In our simple proof-of-concept solution, with the dynamic storage configurations, we are able to save from 21% to 52% of the total consumption. We attribute this to the performance impact of the storage on other components of the phone. We suspect that the interesting savings are triggered by the changes in the storage, and further propagated into other components in the phone. This naturally raises a question, how storage affects the performance of different smartphone subsystems. Thus, still more research needs to be done in this matter.

Acknowledgements

This work is supported in part by NSF grant CNS-1250180. The author would like to thank Dr. Zhou, William & Mary LENS research lab members, and anonymous reviewers for their valuable comments.

Biographical Sketch

David Nguyen has been working on his Ph.D. in Computer Science at the College of William and Mary since Fall 2011. He is advised by Dr. Gang Zhou, and his research interests are wireless networking and smartphone storage.

References

[1] Block i/o layer tracing: blktrace. *http://linux.die.net/man/8/blktrace*, 2012.

[2] Monsoon power monitor. *http://www.msoon.com*, 2012.

[3] Bovet, D., and Cesati, M. *Understanding the Linux Kernel, Third Edition*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2005.

[4] Brunelle, A. D. *blktrace User Guide*. USA, 2007.

[5] Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *Proc. ATC 2010*, USENIX Assoc. (2010).

[6] Kim, H., Agrawal, N., and Ungureanu, C. Revisiting storage for smartphones. In *Proc. FAST 2012*, USENIX Assoc. (2012).

[7] Nguyen, D. T., Zhou, G., Qi, X., Peng, G., Zhao, J., Nguyen, T., and Le, D. Storage-aware smartphone energy savings. In *Proc. UbiComp 2013*, ACM Press (2013).

[8] Pathak, A., Hu, Y. C., and Zhang, M. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proc. EuroSys 2012*, ACM Press (2012).