Storage-aware Smartphone Energy Savings

David T. Nguyen*, Gang Zhou*, Xin Qi*, Ge Peng*, Jianing Zhao*, Tommy Nguyen†, Duy Le‡
*College of William and Mary, Williamsburg, VA 23185, USA
{dnguyen, gzhou, xqi, gpeng, jzhao}@cs.wm.edu
†Rensselaer Polytechnic Institute, Troy, NY 12180, USA
nguyet11@rpi.edu
‡EMC Isilon, Seattle, WA 98104, USA
duy.le@emc.com

ABSTRACT

In this paper, to our best knowledge, we are first to provide an experimental study on how storage techniques affect power levels in smartphones and introduce energy-efficient approaches to reduce energy consumption. We evaluate power degradation at several layers of block I/O, focusing on the block layer and device driver. At each level, we investigate the amount of energy that can be saved, and use that to design and implement a prototype with optimal energy savings named SmartStorage. The system tracks the run-time I/O pattern of a smartphone that is then matched with the closest pattern from the benchmark table. After having obtained the optimal parameters, it dynamically configures storage parameters to reduce energy consumption. We evaluate our prototype by using the 20 most popular Android applications, and our energy-efficient approaches achieve from 23% to 52% of energy savings compared to using the current techniques.

Author Keywords

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

ACM Classification Keywords

C.5.3 Computer System Implementation: Microcomputers; C.4 Performance of Systems: Design studies

General Terms

Design; Experimentation; Measurement; Performance.

INTRODUCTION

Continual advancements in the technology of smartphones have become an important, if not essential, aspect of our daily life. This is unsurprising since a single mobile device has the ability to call and text family members, check status updates on social media sites, access news and information on the Internet, and play a variety of games for entertainment. However, a common complaint among smartphone owners is the poor battery life. To many such users, being required to charge the smartphone after a single day of moderate usage is unacceptable. In a 2011 market study conducted by ChangeWave [1] concerning smartphone dislikes, 38% of the respondents listed that battery life was their biggest complaint, with other common criticisms such as poor 4G capacity and inadequate screen size lagging far behind. The result of such a study demonstrates the necessity for solutions which address the issue of energy consumption in smartphone devices.

In this paper, we investigate the direct impact of smartphone storage techniques on total energy consumption and we answer two key research questions: *How does storage affect smartphone power efficiency?* and *How can we optimize smartphone storage in order to save more energy?* By answering the first question, we find which and how each storage component contributes to the total energy consumption. Different storage techniques have different effects on application performance that results in varying power levels. That leads to our second research question that helps us find ways on optimizing storage approaches in order to save more energy. Answers to these research questions will help engineers come up with more sophisticated storage designs better tailored to modern smartphones and more efficient savings, affecting as many as 1.038 billion smartphone users around the globe (as of September 2012 [12]).

In order to answer the first research question, we evaluate smartphone power efficiency at various layers of the I/O path, such as the block layer and device driver. We 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 impact the total energy consumption differently. We find for 8 benchmarks the combinations of scheduling algorithms and queue lengths with optimal energy savings. In order to address the second research question, we design our SmartStorage system and implement it on the Android platform. SmartStorage 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.

We found a few works in the research community closely relating to ours. The work of Kim et al. [20] presents an 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. [16]

UbiComp'13, September 8–12, 2013, Zurich, Switzerland.

http://dx.doi.org/10.1145/2493432.2493505

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. [22] 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. Our work is also motivated by cross-layer I/O analysis studied by the authors in [24, 25], which has not been done in smartphones. At this stage, there has been no direct study of the correlation between storage techniques and energy consumption within smartphone devices. We believe our work can help other researchers realize the importance of storage and perhaps trigger more exciting solutions to the smartphone energy consumption problem.

In summary, our contributions within this paper are the following:

- First, we provide an experimental study on how storage techniques impact energy consumption on smartphones.
- Second, we design and implement the SmartStorage system that tracks I/O pattern of smartphones in run-time and dynamically configures storage parameters with optimal energy savings.
- Third, we evaluate our solution with an Android-based smartphone on the 20 top free applications from Android market and show that our system can save from 23% to 52% of energy. This is achieved with 2.5% energy overhead from running SmartStorage and a difference of 3% in terms of application delay.

BACKGROUND AND MOTIVATION

In this section, we introduce background and motivation of our work. Next, we explain the measurement and methodology. Afterwards, we proceed with the measurement on the cache, the block layer, and the device driver. Finally, we give summary of our measurement results.

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 cache, the block layer, and the 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 [14].

Cache

The disk cache is a software mechanism allowing the system to keep in RAM some data normally stored on the disk. Further accesses to that same data can be granted without accessing the disk [14]. There are 2 classical caching policies, *write-back* and *write-through*. Write-back is the default approach

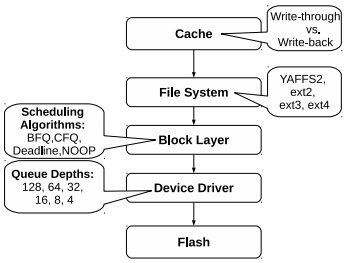

Figure 1. Kernel Components on the I/O Path.

used in smartphones which in practice means that the devices signal I/O completion to the operating system before data has hit the flash disk. In contrast, a write-though cache performs all write operations in parallel, with data written to the cache and the disk simultaneously. There are two mechanisms to control caching behavior of the storage devices, forced cache flush and force unit access. In the Android kernel, write-through is enforced by setting the true value to *REQ_FLUSH* and *REQ_FUA* parameters. If we want the phone to use one of the caching policies, after updating the parameters for the corresponding method in the kernel source code, the kernel needs to be rebuilt and re-flashed into the phone.

File system

There are several file system types used by smartphone vendors, each flash partition can be formatted in a different file system type before being properly mounted to given namespaces such as /data, /system, or /cache. Most frequently used file systems are YAFFS2, ext2, ext3, and ext4. YAFFS2 is used, for instance, in HTC Hero or Google Nexus One. Ext4 is employed in the most recent Android smartphones such as Google Nexus 4 or Samsung Galaxy S4. We can get information regarding the file systems in use by calling *mount* command on a rooted phone.

Block Layer

Block layer is another component on the I/O path. At this level, 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 BFQ (Budget Fair Queuing), each process is assigned a fraction of disk (budget) measured in number of sectors and the disk is granted to a process until the budget expires. CFQ (Complete Fair Queuing) attempts to distribute available I/O bandwidth equally among all I/O requests. The requests are placed into per-process queues where each of the queues gets a time slice allocated. Deadline algorithm attempts to guarantee a start time for a process. The queues are sorted by expiration time of processes. Noop inserts incoming I/Os into a FIFO fashion queue and implements request merging. In some Android phones, the default fixed scheduling algorithm is BFQ (Google Nexus One), others use CFQ (Samsung Galaxy Nexus, Samsung Nexus S).

Device Driver

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., Samsung Galaxy Nexus or Google Nexus One).

Flash

The last level to be reached by the I/Os is the storage subsystem that contains an internal NAND flash memory, an external SD card and a limited amount of RAM. The subsystem contains a number of partitions depending on vendors. The partitions can be found in the /dev/block directory.

Motivation

In this section, the first research question *How does storage affect smartphone energy efficiency?* is addressed by discussing preliminary measurements. We list the chosen benchmarks for our experiments and measure smartphone power level with default I/O path parameters. Afterwards, power level affected by each layer is investigated, starting with the block layer and moving further to the device driver layer. Caching policies are discussed at the end due to lower overall impact. All results are averaged over ten measurements with corresponding confidence intervals.

Benchmark	Properties
AnTuTu [3]	storage, memory, CPU, GPU
CF-Bench [6]	storage, memory, CPU, Java
GLBenchmark [7]	GPU
BrowserMark [5]	browser, JavaScript, HTML
AndroBench [2]	storage, SQLite
Quadrant [9]	storage, memory, CPU, GPU
Smartbench [10]	storage, memory, CPU, GPU
Vellamo [11]	storage, memory, CPU, GPU, browser

Table 1. Benchmarks.

Benchmarks

We run 8 popular benchmarks on the Google Nexus One phone with Android platform under different storage configurations and measure power consumption levels with the Monsoon Power Monitor [8] (details given in Performance Evaluation). Each benchmark tests different phone subsystems and has its specific I/O pattern. The aim is to cover as many I/O pattern types as possible. The 8 chosen benchmarks with their properties are listed in Table 1. We do not use a synthetic benchmark that simulates I/O patterns, since we aim to use application benchmarks that reflect real Android application behavior.

Block Layer Level

The default file system, scheduling algorithm, queue depth, and caching policy for the Google 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. This is an expected outcome due to different I/O request reordering and merging of each scheduling algorithm [24]. Another

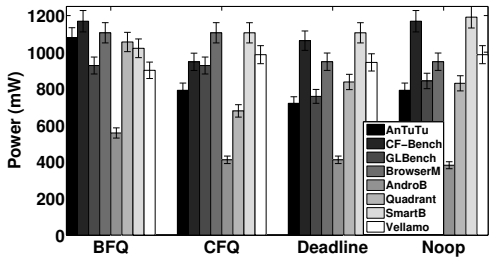

Figure 2. Power for Default Configurations. The measurements are obtained while keeping default *queue depth (128)* and *write-back* caching policy.

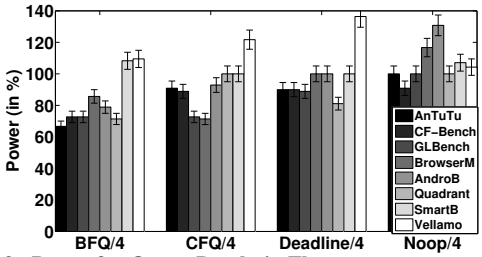

Figure 3. Power for Queue Depth 4. The measurements are obtained while keeping default *write-back* caching policy. The results are compared to the power levels with default configurations in Figure 2 that are baseline values.

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.

Device Driver Level

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 Google 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. This way we can see the potential of how much more power efficient the system will be if we change the queue depth. 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. However, there are some exceptions such as in the case of Smartbench and Vellamo (last two benchmarks) that with smaller queue depth do not perform well and consume on average more power than expected. These two benchmarks are not storage intensive (Table 2), hence, the smaller queue causes higher overhead and as result, the higher consumption is observed.

Cache

This section attempts to find out how power level differs when using write-back and write-through caching approaches. The

Benchmark	Reads Completed/s	Writes Completed/s	RP	Optimal Configuration	Power Savings	Running Time(s)
AnTuTu	1108	1395	0.79	Deadline/4	40%	224.76
CF-Bench	104	1298	0.08	CFQ/4	27.27%	148.18
GLBenchmark	253	51	4.96	Deadline/4	27.27%	254.21
BrowserMark	185	115	1.61	CFQ/4	28.57%	278.14
AndroBench	2260	104	21.73	Noop/128	31.58%	45.1
Quadrant	301	400	0.75	BFQ/4	42.86%	129.11
Smartbench	26	2	13	BFQ/128	0	217.35
Vellamo	9	1	9	BFQ/128	0	49.82

Table 2. Benchmark I/O Patterns. The 2nd column includes the rate of Reads completed per second. The 3rd column includes the rate of Writes completed per second. The 4th column includes *Rate Proportion* (RP), where *RP = Number of Reads Completed per second / Number of Writes Completed per second*. The 5th column includes a combination of a scheduling algorithm and queue depth with optimal power consumption. The 6th column includes power savings of the optimal configuration compared to the default BFQ/128 configuration. The last column is time to complete a benchmark.

phone's consumption with the default caching policy (write-back) can be again determined from Figure 2. Hence, here for each benchmark we measure the power levels with write-through cache (scheduling and queue depth fixed). For easier reading, this is normalized to the power levels with the default write-back cache. Figure 4 shows that write-through caching consumes on average slightly less power. The difference is approximately 10%. This is due to limited queuing buffer space at the disk [24]. If write-back policy is in use, under heavy load the effective queues reach to the maximum allowable value, which is in our case 128. If the buffer queue is full, the device driver delays additional I/O requests. Consequently, that causes the system to slow down and consume more energy. For applications that require more reliability and consistency, write-through cache can be of help. The price for this small improvement in average power consumption requires rebuilding the Android kernel. For this reason, we decide not to include cache layer modifications into our design to provide better scalability and simpler deployment to ordinary users.

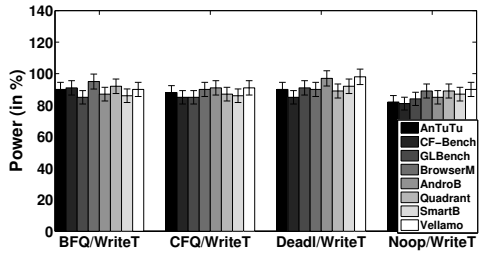

Figure 4. Power for Write-through Cache. The measurements are obtained while keeping default *queue depth (128)*. The results are compared to the power levels with default configurations in Figure 2 that are baseline values.

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. Table 2 shows the final combinations with optimal power consumptions for each benchmark. We can see that Quadrant consumes least power with the combination of BFQ/4, in which case it consumes 754mW and therefore, we save almost 43% compared to the default configuration (BFQ/128).

As with many other optimizations, this significant improvement in power saving has its trade-off that causes performance degradation. In particular, we observe worsen performance of the CPU, GPU, RAM, and I/O. This degradation has to be minimal for users to fully appreciate our proposed solution in the following section. For illustration, the performance scores of AnTuTu benchmark is listed in Table 3 for the default parameters (BFQ/128) and the parameters with optimal power consumption (Deadline/4). The higher benchmark scores, the better performance of a subsystem. For instance, the CPU performance score decreases from 958 to 946 after changing the default parameters to the optimal ones. Similarly, there is a slight GPU performance decrement from 880 to 865. This is expected due to the trade-off from the different I/O ordering and queue depth. However, the average performance degradation is only less than 2%.

Configuration	CPU	GPU	RAM	I/O
Default (BFQ/128)	958	880	317	270
Optimal (Deadline/4)	946	865	317	268

Table 3. AnTuTu Benchmark Performance Scores.

SMARTSTORAGE DESIGN

In order to address the second research question on how to optimize storage to save energy in smartphones, we present SmartStorage. Further in this section, an architecture is introduced and implementation details are discussed.

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.

System Architecture

The architecture is illustrated in Figure 5. It 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 each module and its functionalities.

Kernel Space

SmartStorage Core. This module 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

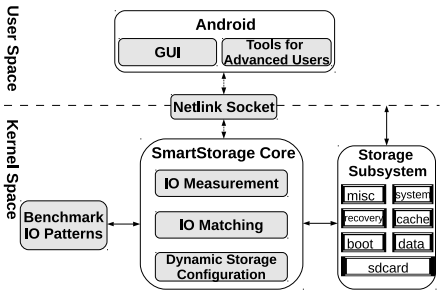

Figure 5. SmartStorage Architecture.

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 [4, 15]. Blktrace is a block layer I/O tracing utility that provides information about request queue operations coming into storage subsystem. Blktrace is normally available in Linux distributions but it needs to be enabled in the Android system. Typically, the blktrace output includes a process ID, type of an I/O such as Read or Write, its time stamp, sequence number, etc. 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 I/O patterns. The I/O 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. We find that this simple matching method is precise. More will be discussed in the Performance Evaluation section. Finally, the optimal scheduling algorithm is set in the block layer scheduler and the optimal queue depth is set in the device driver. This is done on all partitions.

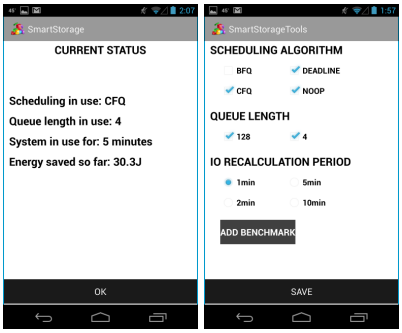

Figure 6. Screen shots of the GUI and Tools for Advanced Users.

Benchmark I/O Patterns. This includes a table with our benchmarks and their I/O patterns. See Table 2. Each benchmark is paired with a combination of a scheduling algorithm and queue depth with optimal power consumption obtained offline. The I/O pattern consists of rates of a specific type of I/O per second: Number of Reads completed per second and Number of Writes completed per second.

User Space

SmartStorage works naturally as a background service to save energy, without any need of interaction with users. It does not require popping up GUI or the Tools for Advanced Users. The two additional components are there only for convenience of interested users. The screen shots of the GUI and Tools for Advanced Users are included in Figure 6. We describe these two components below.

GUI. The graphic user interface provides the current status of the system. That includes information on which scheduling algorithm and queue depth are being used. It also informs of how long the system is being in use and how much energy has been saved.

Tools for Advanced Users. This part is designed to serve researchers and advanced users who can possibly contribute to further project advancements. The tools let users edit after how long system should recalculate I/O pattern of the phone. It also allows setting preferred scheduling algorithms and queue lengths to be considered in dynamic configurations. With scalability in mind, it is possible to add new benchmarks to the Benchmark I/O Patterns table in the future. Finally, it communicates preferences to the SmartStorage Core in kernel space through a netlink socket [23].

Implementation

We implement our system on the HTC Google Nexus One smartphone with Android 2.3.7 and kernel 2.6.37.6. The implementation has 2 main parts, SmartStorage Core that is in the kernel space and the GUI that is developed as an application. As mentioned earlier, SmartStorage works naturally as a background service to save energy, without any need of interaction from the users. The additional user interface is only for the convenience of the researchers and interested users. This section highlights some of the important implementation details of the SmartStorage Core and the GUI.

SmartStorage Core is implemented as a kernel module. It interacts with the application layer through a netlink socket.

It sends up information regarding the current status, including the combination of a scheduling algorithm and queue depth in use. That scheduling algorithm is obtained by reading from a block layer scheduler. For instance, following we obtain the algorithm in use in the /data partition (mtdblock5): *cat /sys/block/mtdblock5/queue/scheduler*. Similarly, the queue depth in use in the /data partition is obtained from the nr_requests (number of requests) parameter:
cat /sys/block/mtdblock5/queue/nr_requests.

SmartStorage calculates the phone's I/O pattern periodically each minute. After having found an optimal combination of a scheduling algorithm and queue depth via matching the phone's I/O pattern to the benchmark I/O pattern table explained earlier, the system enforces the use of the scheduling algorithm and queue depth found. For example, to set CFQ for the /data partition, the scheduler file of the block device is modified on-the-fly as follows:
$echo\ cfq\ >\ /sys/block/mtdblock5/queue/scheduler$.
The queue depth of the /data partition can be changed on-the-fly to 4 by modifying its nr_requests parameter:
$echo\ 4 > /sys/block/mtdblock5/queue/nr\_requests$.

In order to use blktrace, the support for tracing block I/O actions is enabled by changing the menuconfig file to support tracing block I/O actions. Afterwards, the makefiles are modified to include blktrace.

PERFORMANCE EVALUATION

This section evaluates the SmartStorage solution by a series of comprehensive experiments and answering the following questions. The first two questions are related to energy savings: *(1) How does SmartStorage save energy with a typical use case?* We address this by comparing energy usage of the 20 most popular applications from the Android Market with and without SmartStorage. *(2) What is the overhead of SmartStorage?* We address this by measuring overhead in energy usage of SmartStorage compared to the case when it is not in use. The last two questions are related to performance issues: *(3) How precisely does SmartStorage match I/O patterns to storage configurations with optimal energy savings?* Here we show how many applications get matched with the correct combinations. *(4) Does our SmartStorage solution incur performance penalties?* This is determined using the AndroBench benchmarking tool testing throughput and I/O performance. In addition, we run ten most popular applications and evaluate their application delays.

Experiment Setup

In our experiments, we use the SmartStorage implementation in the HTC Google Nexus One phone. To measure energy consumption, the Monsoon Power Monitor [8] is utilized. We run the experiments with the top 20 free applications from the Android Market as of August 7, 2012. See Table 4. The Monsoon Power Monitor is configured by blocking the positive terminal on the phone's battery with electrical tape. The voltage normally supplied by the battery is supplied by the monitor. It records voltage and current with a sample rate of 5 kHz. 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.

Top 20 Apps	RP	SmartStorage Combination	Optimal Combination
#1 App	2.6	CFQ/4	CFQ/4
#2 App	0.09	CFQ/4	CFQ/4
#3 App	0.28	CFQ/4	CFQ/4
#4 App	14.7	BFQ/128	BFQ/128
#5 App	12.29	BFQ/4	BFQ/4
#6 App	4	Deadline/4	Deadline/4
#7 App	*9.09*	*BFQ/4*	*BFQ/128*
#8 App	0.79	Deadline/4	Deadline/4
#9 App	0.24	CFQ/4	CFQ/4
#10 App	1.36	CFQ/4	CFQ/4
#11 App	1.27	CFQ/4	CFQ/4
#12 App	0.28	CFQ/4	CFQ/4
#13 App	2.03	CFQ/4	CFQ/4
#14 App	11.2	BFQ/128	BFQ/128
#15 App	0.02	CFQ/4	CFQ/4
#16 App	2.5	CFQ/4	CFQ/4
#17 App	0.26	CFQ/4	CFQ/4
#18 App	0	CFQ/4	CFQ/4
#19 App	0.81	Deadline/4	Deadline/4
#20 App	4.94	Deadline/4	Deadline/4

Table 4. The 20 applications used in evaluation. The top 20 free applications from the Android Market as of August 7, 2012: **#1.Gmail, #2.YouTube, #3.Facebook, #4.Lookout Security, #5.Google Maps, #6.Twitter, #7.Tiny Flashlight + LED, #8.Yelp, #9.Amazon MP3, #10.Tango Video Calls, #11.Temple Run, #12.WhatsApp Messenger, #13.Adobe Flash Player, #14.Instagram, #15.Google Play Books, #16.Pandora Internet Radio, #17.ColorNote Notepad Notes, #18.Amazon Mobile, #19.GO SMS Pro, and #20.Voice Search.** The second column includes *Rate Proportion* (RP), where *RP = Number of Reads Completed per second / Number of Writes Completed per second*. The third column includes the combination of scheduling algorithm and queue depth determined by SmartStorage. The last column includes the optimal combination. 19 from 20 applications (95%) are matched with the correct combination by SmartStorage. Tiny Flashlight + LED (7.) is matched incorrectly.

Energy Savings

As far as energy saving is concerned, it is our priority to save as much as possible, and at the same time, the system should not cause significant overhead. We show our results in this subsection.

Energy Savings

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 7. 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.

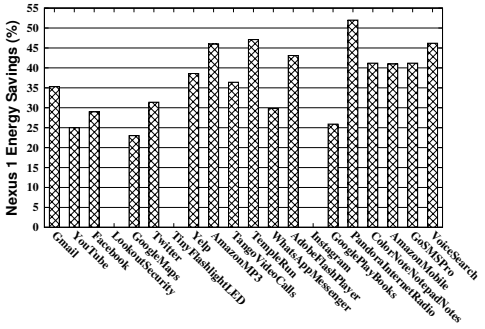

Figure 7. Energy Savings on Nexus One. Energy savings with SmartStorage compared to the use with default scheduling algorithm and queue depth (BFQ/128)

As for validation, we also deploy our solution on the LG Google Nexus 4 smartphone with the latest Android OS 4.2.2 (Jelly Bean), kernel 3.4, and Ext4 file system. The results of the total savings on the Nexus 4 phone are in Figure 8. Gmail, YouTube, Facebook, and other applications with the optimal scheduling algorithm CFQ have slightly less energy savings on Nexus 4. That is expected, since the Nexus 4 phone has the default configuration CFQ/128, hence, additional savings are low.

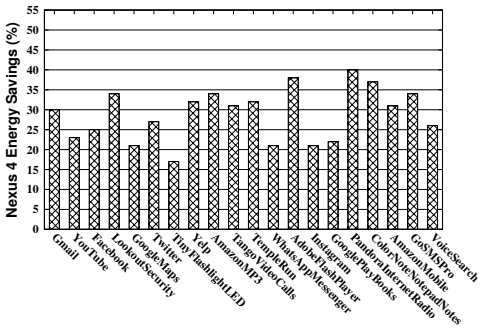

Figure 8. Energy Savings on Nexus 4. Energy savings with SmartStorage compared to the use with default scheduling algorithm and queue depth (CFQ/128)

Real-time Power Consumption

To show how SmartStorage saves energy in real-time, we conduct the following experiment. We utilize the Monkey tool to use five applications (YouTube, Pandora, Google Maps, Amazon Mobile, and Facebook) over 20 minutes (app per four minutes). This is done two times, once with SmartStorage enabled, the other time with SmartStorage disabled. Both times we make sure that the application cache is cleared before a new run. Monkey is run with the same seed, hence triggering the same series of events both times. This assures we get precise results without any delay caused by inconsistent user behavior. We use Monkey to launch YouTube and watch a video clip, the total time including both loading and watching is four minutes. After having closed YouTube (closing time negligible), we load Pandora and listen to a channel, again the total time is four minutes. Similarly, our Monkey script continues with Google Maps to find a few spots, loads Amazon Mobile to search several products, and finally uses Facebook to check the news feed.

The resulting power levels logged from the monitor are in Figure 9. Each four-minute segment shows data of an application. We can observe that after approximately two minutes of use of an application, the phone's power level drops. This is an expected behavior. Since the I/O pattern recalculation is done after each minute, and the loading time of an application takes a while, we should be looking for significant power drop after approximately two minutes. For YouTube (minute 0-4), the first spikes load the application, then the power reaches 2000 mW when SmartStorage is disabled. When SmartStorage is enabled, the values drop to around 50% after two minutes of use. Pandora (minute 4-8) has the power dropping after a little later than expected because an advertisement pops up when SmartStorage is enabled. For Google Maps (minute 8-12), the number of high power spikes significantly decreases after slightly more than two minutes. When using Amazon Mobile (minute 12-16), even earlier than 14th minute, power drops to approximately 50%. Finally, Facebook (minute 16-20) shows a similar behavior, power level goes down during the 18th minute. The power levels here, however, are varying slightly more. We attribute this to the difficulty to replicate the exact behavior on Facebook that has more features that can affect the power levels than the other ones.

The total energy saving in this experiment is 152.3J which corresponds to 31.5% of the total energy consumption. In theory, according to the results in Figure 7, if we run each application separated, we will get 34% saving in total by averaging the savings of all five applications. The difference of 2.5% we account to the recalculation period of one minute that is not optimal. We discuss possible improvements of this approach later in Future Work.

Cost

To address the energy overhead, we run the 5 applications above separatedly with SmartStorage enabled and measure the energy consumption. These applications are running again with SmartStorage disabled and the measurement is repeated. Finally, the 2 consumptions are subtracted and the overhead is obtained. The average overhead is 2.6% of energy consumption for Pandora, and 2.5% for other four applications. This is negligible considering the fact that with SmartStorage enabled we can save from 23% to 52% of energy consumption including the cost mentioned.

Performance

First, we are interested in the I/O pattern matching performance. Next, we ensure that SmartStorage incurs small performance penalties. Finally, we provide a discussion on application delays.

I/O Pattern Matching

The results of I/O matching are presented in Table 4. We can see the results of the final combinations of a scheduling algorithm and queue depth found by SmartStorage. In 19 cases out of 20, the matching method works correctly, which

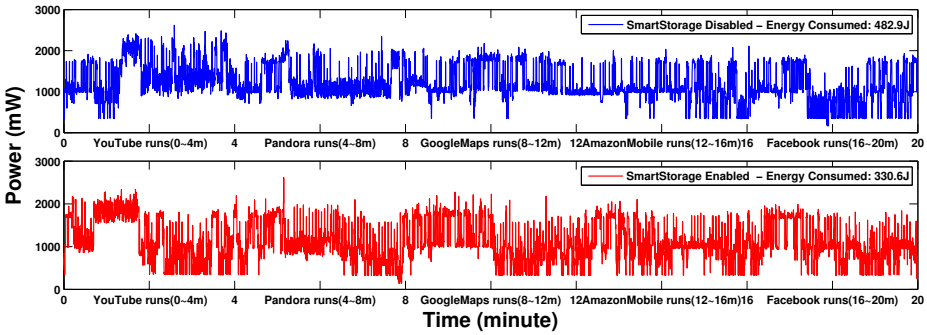

Figure 9. Real-time Power. The figure shows two cases, when SmartStorage is disabled (top), and when SmartStorage is enabled (bottom). We run in order five applications, YouTube (minute 0-4), Pandora (minute 4-8), Google Maps (minute 8-12), Amazon Mobile (minute 12-16), and Facebook (minute 16-20). The time includes both loading application and the use itself. Since our algorithm recalculates the I/O pattern periodically every minute, and the loading time also takes a while, power drops after around two minutes of use of each application as in the figure is expected.

presents 95% accuracy. We note that there are 8 configurations (4 scheduling algorithm choices multiplying 2 queue length choices), each of which is the potential optimal configuration for some applications. However, the 8 selected benchmarks only cover 5 of them. Although the current I/O pattern matching method is precise for the 20 most popular applications we evaluate, in future we plan to explore a machine learning based method. More is discussed in the Future Work.

Performance Penalties

The AndroBench benchmarking tool [2] is utilized to evaluate the performance penalties of SmartStorage, since it heavily loads storage and provides flexible workload settings. The experiments are run on the Google Nexus One phone with 165MB of free space in the internal storage. We run the benchmark with four different workloads: a read-dominated, write-dominated, and a balanced workload of reads and writes of different working set sizes (large and small). Each workload is run for the case when SmartStorage is enabled, and the case when it is disabled. The benchmark throughput in transactions per second (TPS) is measured, followed by the number of disk requests completed in each second (IOPS).

The used benchmark specifications are listed in Table 5. For each workload we set the read file size, write file size, and number of transactions. We aim to have significant difference between the read file size and the write file size for Read-dominated and Write-dominated workloads. In our case, one size is 16 times larger than the other one. Furthermore, we differentiate the large workload, i.e., occupying approximately 40% of free space, and the small workload, i.e., occupying a few percentage of free space. The number of transactions is set to a constant.

Workload	Read Size	Write Size	Transactions
Read-dominated	32MB	2MB	300
Write-dominated	2MB	32MB	300
Balanced Large	32MB	32MB	300
Balanced Small	2MB	2MB	300

Table 5. Workload Parameters.

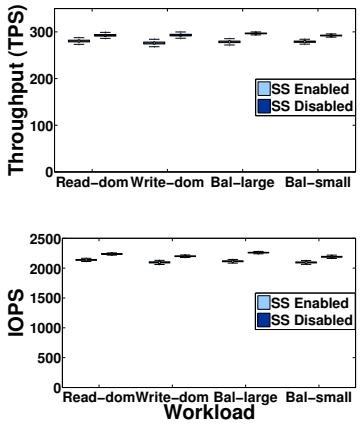

Figure 10. SmartStorage Throughput and I/O Performance.

Each workload is run twice, once with SmartStorage enabled and once disabled. Figure 10 shows the benchmark throughput in number of transactions per second for both runs. This indicates negligible difference between the 2 cases. The throughput varies from 4% to 6%. The biggest penalties are for the case of write-dominated workload and balanced large workload.

The second part of Figure 10 illustrates the disk I/O performance for the case with SmartStorage enabled and the case with SmartStorage disabled. The performance in number of disk I/Os completed per second is demonstrated for all four workloads mentioned. Similar to the previous case, the performance penalties vary from 4% to 6%. The biggest penalty comes with the large workload of balanced reads and writes. This is another spot where the performance can possibly be improved. As in the previous case with the number of transactions per second, the large workload seems to be problematic, this can be a key to research how to further optimize the system performance.

Application Delay

While saving energy is important, having solid performance with small application delays is equally important. In order to test delays of applications running on the phone with

SmartStorage, we utilize the Android Monkey tool. Using it, we generate pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. We run the experiments with the 10 most popular Android applications on the Google Nexus One smartphone with SmartStorage enabled, and the second time with SmartStorage disabled. Each application has a predefined set of user activities triggered through the Monkey tool. We measure the time delay for both cases, when SmartStorage is enabled, and when it is disabled.

Monkey is a command-line tool that can send a stream of events into the phone's system in a random yet repeatable manner. Each series of events we specify with the same seed value (10) in order to generate the same sequence of events. We insert a fixed delay between events (1000 ms) and adjust percentage of different types of events. We fix the number of events as a constant (500). The events are individually adjusted for each application to represent a typical usage, for instance, in Gmail we read and write an email, add a contact, change a label, etc. We run the experiments with the 10 most popular applications from Table 4, once with SmartStorage enabled, the next time with SmartStorage disabled. Each time we output the time delay in milliseconds and the results are illustrated in Figure 11. We can see that the difference is less than 3%, thus we can claim with confidence that the application delays caused by SmartStorage are negligible.

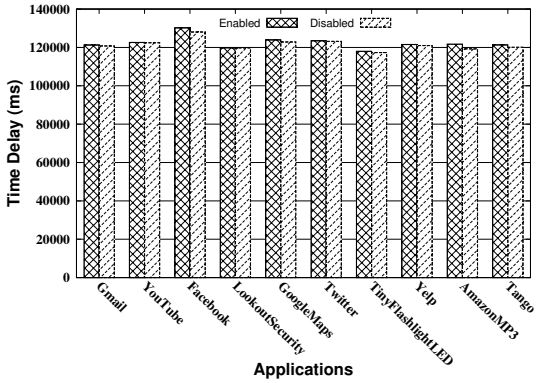

Figure 11. Application Delay.

RELATED WORK

We divide related work into 3 categories: smartphone storage performance, smartphone power consumption analysis, and I/O optimization.

Smartphone storage performance. Little work has been done to examine the storage performance in smartphones. Kim et al. [19, 20] discover a strong correlation between storage and application performance degradation. Their discovery of the correlation highlights the importance of the storage performance. We take a deep look into several components of the I/O path from the application layer to the storage and exploit energy savings at the block layer and the device driver. Thus, we consider the above works as complementary to ours.

Smartphone power consumption analysis. Works have been done to analyze the power consumption of network traffic in smartphones. Gupta et al. [17] measure the WiFi power consumption by various network activities in smartphones. Balasubramanian et al. [13] measure the power consumption characteristics of GSM, WiFi, and 3G. In this paper, we focus on the power consumption analysis of the storage system in smartphones. Several works measure the power consumption of different components in the smartphone. Carroll et al. [16] presents a detailed power consumption analysis of different smartphone subsystems. However, the smartphone used is only for scientific purpose rather than practical usage. In [21, 18], the authors measure and model the power consumption of several hardware subsystems, including CPU, display, graphics, GPS, audio, microphone, and WiFi. In contrast, our work focuses on investigating the impact of the storage system on energy efficiency.

I/O optimization. Riska et al. [24] evaluate the I/O performance improvement of optimization at various layers (system layer, device driver layer, and disk driver layer). Shin et al. [25] propose two new techniques, request bridging and request interleaving, to improve the I/O performance of small writes. However, this group of work focuses only on conventional computer systems that require significantly different optimization techniques.

CONCLUSIONS AND FUTURE WORK

In this paper, we presented an experimental study of how storage parameters in the cache, device driver, and block layer affect the power levels of mobile devices running Android. In addition, we proposed a system called SmartStorage that dynamically tunes storage parameters to reduce energy consumption by matching the current I/O pattern to a known pattern that we recorded from the eight benchmarks. Finally, we validated our dynamic tuning technique by showing that SmartStorage saved 23% to 52% of the energy consumption by running SmartStorage in the background with selected applications from the top 20 most popular apps in the foreground.

In this work, we noticed that the write-back policy consumes on average 10% more energy than write-through with the eight selected benchmarks. This could further increase our energy savings. However, this knowledge was not used in our implementation because switching from write-back to write-through or vice-versa would require rebuilding the kernel. That would be impractical for future deployment. Similarly, changing the file system and copying data around is counter-productive but could provide additional energy savings when considering the overall interaction with the remaining storage components and their parameters. In the future, we will model the trade-off between energy savings and performance degradation with write-back and write-through, and analyze how the file system interacts with the remaining components to understand and explore additional energy savings. 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 since what matters most is the num-

ber of writes and reads in a time interval and not the ordering of them. We plan to research the following machine learning based method in the future. In the method, each configuration is considered as a target class. We plan to collect I/O patterns from a large number of applications and label the I/O pattern of each application as its optimal configuration class. The optimal configurations of all applications should cover all 8 choices. With this data, we train either a supervised classification model or an unsupervised clustering model for run-time I/O pattern matching. 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 energy consumption [16]. Our system with dynamic storage configurations saves from 23% to 52% of the total energy consumption. We need to emphasize that these savings are the savings of the whole smartphone, not only of the storage subsystem itself. We attribute this to the performance impact of storage on other phone components. We suspect that the interesting savings are triggered by the changes in storage, and further propagated into other components in the phone. This opens a new research question, and that is, how storage affects the performance of different smartphone subsystems. Kim et al. [20] already show how performance of smartphone applications is affected by storage performance, but do not consider energy performance. Therefore, still more research is required and we hope that our results will motivate a deeper look into this exciting area.

ACKNOWLEDGEMENT

This work is supported in part by NSF grant CNS-1250180. We thank William & Mary LENS research lab members and anonymous reviewers for their valuable comments.

REFERENCES

1. Changewave research. *http://www.changewaveresearch.com*, 2011.
2. Androbench benchmark. *http://www.androbench.org/wiki/AndroBench*, 2012.
3. Antutu benchmark. *http://www.antutu.com*, 2012.
4. Block i/o layer tracing: blktrace. *http://linux.die.net/man/8/blktrace*, 2012.
5. Browsermark benchmark. *http://browsermark.rightware.com/*, 2012.
6. Cf-bench benchmark. *http://bench.chainfire.eu/*, 2012.
7. Glbenchmark. *http://www.glbenchmark.com/*, 2012.
8. Monsoon power monitor. *http://www.msoon.com*, 2012.
9. Quadrant benchmark. *http://www.aurorasoftworks.com/products/quadrant*, 2012.
10. Smartbench benchmark. *http://www.1mobile.com/smartbench-2012-327800.html*, 2012.
11. Vellamo benchmark. *http://www.quicinc.com/vellamo/*, 2012.
12. Worldwide smartphone users cross 1 billion mark. *http://www.ibtimes.com*, 2012.
13. Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. IMC 2009*, ACM Press (2009).
14. Bovet, D., and Cesati, M. *Understanding the Linux Kernel, Third Edition*, 3 ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2005.
15. Brunelle, A. D. *blktrace User Guide*. USA, 2007.
16. Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *Proc. ATC 2010*, USENIX Assoc. (2010).
17. Gupta, A., and Mohapatra, P. Energy consumption and conservation in wifi based phones: A measurement-based study. In *Proc. SECON 2007*, IEEE (2007).
18. Jung, W., Kang, C., Yoon, C., Kim, D., and Cha, H. Nonintrusive and online power analysis for smartphone hardware components. Tech. rep., 2012.
19. Kim, H., Agrawal, N., and Ungureanu, C. Examining storage performance on mobile devices. In *Proc. MobiHeld 2011*, ACM Press (2011).
20. Kim, H., Agrawal, N., and Ungureanu, C. Revisiting storage for smartphones. In *Proc. FAST 2012*, USENIX Assoc. (2012).
21. Murmuria, R., Medsger, J., Stavrou, A., and Voas, J. Mobile application and device power usage measurements. In *Proc. SERE 2012*, IEEE (2012).
22. 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).
23. Pyles, A., Qi, X., Zhou, G., Keally, M., and Liu, X. Sapsm: Smart adaptive 802.11 psm for smartphones. In *Proc. UbiComp 2012*, ACM Press (2012).
24. Riska, A., Larkby-Lahet, J., and Riedel, E. Evaluating block-level optimization through the io path. In *Proc. ATC 2007*, USENIX Assoc. (2007).
25. Shin, D., Yu, Y., Kim, H., Eom, H., and Yeom, H. Request bridging and interleaving: Improving the performance of small synchronous updates under seek-optimizing disk subsystems. *ACM Transactions on Storage (TOS)* (2011).