Examining Storage Performance on Mobile Devices

Hyojun Kim[*]
Georgia Institute of Technology
hyojun.kim@cc.gatech.edu

Nitin Agrawal, Cristian Ungureanu
NEC Laboratories America
{nitin, cristian}@nec-labs.com

ABSTRACT

Conventional wisdom holds that storage is not a big contributor to application performance or energy consumption on mobile devices. Flash storage (the type most commonly used today) draws little power, and its performance is thought to exceed that of the network subsystem. In this paper we present initial evidence to the contrary even for common applications such as web browsing or application install. We find that just by varying the underlying flash storage, performance of web browsing over WiFi can vary roughly by 500%, and of application install by 300%. With a faster network (setup over USB), storage is taxed even more and the performance variation rose to roughly 700% for web browsing! The performance variation can be attributed to the characteristics of the storage device, the workload pattern (random or sequential), and the operating system itself. We also find that lower storage performance leads to increased CPU consumption, thus having an indirect impact on energy.

1. INTRODUCTION

Mobile phones, tablets, and ultra-portable laptops are no longer viewed as the wimpy siblings of the personal computer; for many users they have become the predominant computing device for a wide variety of applications. According to a recent Gartner report, within the next three years, mobile devices will surpass the PC as the most common web access device worldwide [21]. By 2013, over 40% of the enhanced phone installed-base will be equipped with advanced browsers [27].

[*]Work done as an intern at NEC Labs

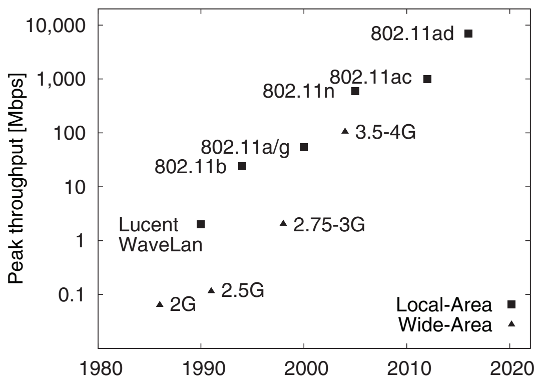


Figure 1: Peak throughput of local-area and wide-area networks for the past three decades; y-axis is log scale.

Research pertaining to mobile devices can be broadly split into applications and services, device architecture, and operating systems. From a systems perspective, recent research has tackled many important aspects: understanding and improving battery life [16], network middleware [25]; application execution models [18, 17]; security and privacy [19, 15, 22]. However, one important component is conspicuously missing from the research landscape – storage. Storage has traditionally not been viewed as a critical component of phones, tablets, and PDAs; at least in terms of the performance expected, as long as its performance exceeds that of the network's, it is difficult for storage to be the bottleneck. We find evidence to the contrary.

Mobile flash storage is typically classified according to *sequential* throughput, yet many commonly used applications on mobile devices exhibit *random* accesses, for which storage performance is significantly worse. The disparity between sequential and random write performance is inherent with flash-based storage.

We believe that impact of storage on performance will continue to increase because of the following trends. First, emerging wireless technologies such as 802.11n (600 Mbps peak throughput) and 802.11ad (7 Gbps peak throughput) offer the potential for significantly faster network connectivity to mobile devices; Figure 1 presents the trends for network performance. Second, mobile devices are being used as the primary computing device, running more performance intensive tasks than previously imagined. In developing economies, a mobile/enhanced phone is often the only computing device owned by an individual and needs to cater to a variety of usage scenarios; smartphone usage is also on the rise.

This work is an attempt to understand the contribution of the storage subsystem to application performance on mobile devices. Many of the applications that are commonly used to browse the web, stream video, sync/backup the phone, make use of the storage subsystem. Due to space constraints, we focus on only two cases: web browsing and application install. Not only are these activities available on all smart phones, but they are done frequently enough that performance problems with them negatively impacts user experience.

We make the following observations through a measurement study of these applications on a set of Android smartphones. First, storage adversely affects their performance, in spite of the fact that they are not thought of as being storage I/O intensive. A recent study of browser performance on smart phones [29] does not even mention storage as a potential issue. For applications that are more I/O intensive, the impact is likely higher. Second, our benchmarking reveals that the "speed class" marking on SD cards is not necessarily indicative of application performance; we find cases in which higher-grade SD cards performed worse than lower-grade ones. Finally, we also observe higher total CPU consumption for the same activity when using lower performance cards. This indicates potential problems with the storage stack on the device, or the application. Unless resolved, lower performing storage not only makes the application run slower, it also increases the device's energy consumption. We are further investigating both the OS and the application software to understand these effects.

Based on our experimental findings and observations, we believe improvements in the storage stack are required along multiple dimensions. First, hardware improvements to the internal storage and the SD card can alone account for improvements to application performance. Device manufacturers like Samsung and Sandisk are actively looking to bring faster devices to the mobile market; Samsung announced the launch of a PCM-based multi-chip package for mobile handsets [28]. Second, mobile I/O and memory bus technology needs to evolve as well to sustain higher throughput to the devices. Third, and perhaps more important from the perspective of the systems community, changes are warranted in the mobile software stack to complement the hardware improvements.

2. MOBILE DEVICE STORAGE SETUP

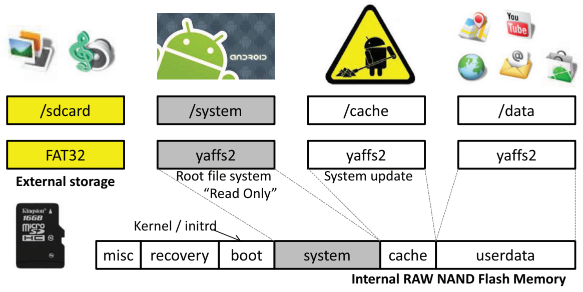


Figure 2: Overview of Android's Storage Schema.

Partition	Function
misc	Miscellaneous system settings (*e.g.*, Carrier ID, USB config, hardware settings, IMEI number); persistent shared space for OS and bootloader to communicate
recovery	Alternative boot-into-recovery partition for advanced recovery and maintenance ops
boot	Enables the phone to boot, includes the bootloader and kernel
system	Contains the remaining OS, pre-installed system applications, and Android user interface; typically a read-only partition
cache	Android can use it to stage and apply "over the air" updates; holds system images
data	Stores user data (*e.g.*, contacts, messages, settings) and installed applications; SQLite database containing app data also stored here. Factory reset wipes this partition
sdcard	External SD card partition to store media, documents, backup files etc
sd-ext	Additional partition on SD card that can act as `data` partition, setup is possible through a custom ROM and `data2SD` software; non-standard Android partition

Table 1: Data storage partitions for Android on HTC Desire; first group represents internal flash partitions and second group represents external partitions.

2.1 Android Storage Subsystem

Most mobile devices are provisioned with a limited amount of RAM, internal flash storage, and an external SD card slot. In addition, some devices (*e.g.*, LG G2X phone) also have a non-removable SD card inside the phone; such storage is still treated as external storage.

The internal flash storage contains all the important system partitions, including partitions for bootloader and kernel, recovery, system settings, pre-installed system applications, and user-installed application data. The external storage is primarily used for storing user content such as media files (*i.e.*, songs, movies, and photographs), documents, and backup images. Figure 2 shows the SD card storage and the internal raw NAND flash storage on an HTC Desire phone. Table 1 presents the functionality of the partitions in detail; this storage

SD Card	Speed	Cost	Performance (MB/s)			
All 16GB	Class	US $	Sq R	Sq W	Rn R	Rn W
Kingston	2	28.71	18.20	6.80	2.76	0.02
Sandisk	4	23.25	13.25	5.61	1.09	0.70
A-Data	6	29.95	18.20	11.04	3.04	0.01
Patriot	10	34.99	18.10	10.79	3.03	0.01

Table 2: Raw device performance and cost. "Sq" is sequential and "Rn" is random performance

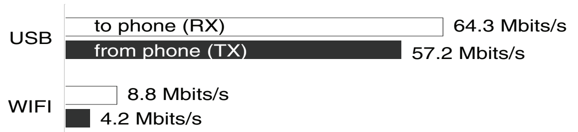


Figure 3: Performance comparison of slow (wifi) and fast (miniUSB reverse tether) networks with iperf.

setup is fairly typical across Android devices.

Applications can store configuration and data on the device's internal storage as well as on the external SD card. Android uses SQLite [14] database as the primary means for storage of structured data. Applications are provided with a well defined interface to create, query, and manage their own databases; one or more SQLite databases are stored per application on the `/data` partition. Android provides a filesystem-like interface to access the external storage. In practice, only a subset of applications use the SD card to store data.

Apple's iOS operating system also uses SQLite to store application data. iOS Core Data is a data model framework built on top of SQLite; it provides applications access to common functionality such as save, restore, undo and redo. iOS 4 does not have a central file storage architecture, rather every file is stored within the context of an application. We focus on Android, since it allows systems-level development.

2.2 Methodology of Phone Setup

In this paper we present experiments conducted on the Google Nexus One [9]. We have also performed the same experiments on the HTC Desire [10], obtaining similar results; we omit their presentation due to space constraints. The Nexus One is a GSM phone with a 1 GHz Qualcomm QSD8250 Snapdragon processor, 512 MB RAM, and 512 MB internal flash storage; the Desire has the same hardware configuration except its RAM is slightly larger at 576 MB. Both phones are running Android Gingerbread 2.3.4, the CyanogenMod 7.1.0 custom firmware [7], and a Linux kernel 2.6.37.6 modified to provide resource usage information. We present a brief description of the generic OS customizations, which are fairly typical, and then explain the storage-specific customization later in this section.

In order to prepare the phones for our experiments, we setup the Android Debug Bridge (ADB) [1] on a Linux machine running Ubuntu 10.10. We subsequently *root* the device with unrevoked3 [12] to flash a custom recovery image (ClockworkMod [4]).

For our experiments we needed to bypass some of the constraints of the stock firmware; for this purpose we install CyanogenMod [7]. In particular, we needed support for *reverse tethering* the mobile device via USB, the ability to custom partition the storage, and access to a wider range of system tools and Linux utilities for development (*e.g.*, BusyBox [3]).

An important requirement, specific to our storage experiments, is to be able to compare and contrast application performance on different storage devices. Some of these applications heavily use the internal non-removable storage. In order to observe and measure all I/O activity, we change Android's `init` process to mount the different internal partitions on the external storage. Our approach is similar to the one taken by Data2SD [11]; in addition, we were able to also migrate to the SD card the `/system` and `/cache` partitions. In order to adhere to Android's boot-time compatibility tests, we provided a one GB FAT32 partition at the beginning of the SD card, mounted as `/sdcard`; the /system, /cache, and /data partitions were formatted as Ext3. Note that this setup is not normally employed by end-users, but it allows us to run what-if scenarios with storage devices of different performance characteristics, of which the internal flash is only one data point.

As part of our experiments, we want to understand the impact of storage on application performance under current WiFi networks, as well as under faster network connectivity (likely to be available in the future). For WiFi, we set up a dedicated wireless access point (IEEE 802.11 b/g) on a Dell laptop. Since we do not have a faster wireless network on the phone, we emulate one by reverse tethering [13] it over the miniUSB cable connection with the same laptop (this allows the device to access the internet connection of the host); Figure 3 compares the performance of the WiFi and the miniUSB RT link using *iperf* [24].

We conducted all experiments on the internal non-removable flash storage and four removable microSDHC cards, one each from the different SD speed classes. Table 2 lists the SD cards[1] along with their specifications and a baseline performance measurement done on a Transcend TS-RDP8K card reader using the CrystalDiskMark benchmark [6], version 3.0.1. Testing size is 100 MB, random I/O size is 4KB, and we report average performance over 3 runs; observed standard deviation is low and we omit it from the table. Prices shown are as ordered from Amazon.com and its resellers, and Buy.com (to be treated as approximate).

To summarize, read performance of the different cards is not a differentiating factor, and much better overall than the write performance. Within writes, Kingston

[1]Note that internal flash could not be measured this way.

performs poorly for random and sequential, A-Data and Patriot perform poorly for random, but well for sequential, and finally, Sandisk performs comparatively much better for random, and poorly for sequential.

3. PERFORMANCE EVALUATION

3.1 Evaluation Methodology

We first explain our measurement environment and the changes introduced to collect performance statistics: (1) We made small changes to the microSD card driver to allow us to check "busyness" of the storage device by polling the status of the /proc/storage_usage file. (2) We wrote a background monitoring tool to periodically read the proc file system and store summary information to a log file; the log file is written to the internal `/cache` partition to avoid influencing the SD card performance. CPU, memory, storage, and network utilization information is obtained from /proc/stat, /proc/meminfo, /proc/storage_usage (busyness) and /proc/diskstats, and /proc/net/dev respectively. (3) We use the standard `blktrace` [2] mechanism to collect block-level traces for device I/O.

In order to ascertain the overheads of our instrumentation, we conducted experiments with and without the measurement environment; we found that our changes introduce an overhead of less than 2% in total runtime. The numbers are not shown due to space constraints.

We now describe in more detail the two Android applications that we use to assess storage's impact on application performance.

WebBench: is a custom benchmark program we wrote to measure web browsing performance in a non-interactive manner; it is based on the standard WebView Java Class provided by Android. WebBench visits a pre-configured set of web sites one after the other and reports the total elapsed time for loading the web pages. In order to accurately measure the completion time, we made use of the public method of WebView class named `onProgressChanged()`; when a web page is fully loaded, WebBench starts loading the next web page on the list. We ran WebBench to visit the top 50 web sites according to a recent ranking [5].

AppInstall: is a benchmark that installs a set of 10 Android applications, successively, using the `adb install` command. The applications, top 10 apps on Google Android Market, are listed in Table 3.

3.2 Results

All the results presented in this section are actual measurements on the mobile phones. We present the average and standard deviation over three runs.

The first set of experiments compare the performance of the two benchmarks on the five different storage devices described earlier. Figure 4(a) shows the runtime of WebBench for WiFi and USB reverse tethering; the y-axis is the total runtime. Surprisingly, even with a relatively slow WiFi network connection, we notice a 500% difference between the best case (Sandisk) and the worst case (Kingston). As expected, the faster the network (USB in RT) the higher the impact of storage: 686% difference between the best and worst cases. Figure 4(b) shows the results of the AppInstall experiment, and we find a similar trend.

Rank	App Name	Version	Size (MB)
1	KakaoTalk	2.1.1	1.56
2	Google Maps	5.5.0	6.09
3	Flash Player	10.3.185	4.22
4	Facebook for Android	1.5.4	2.76
5	YouTube	2.1.6	0.80
6	Pandora Internet Radio	1.5.8	1.22
7	Horoscope	1.5.2	2.95
8	WhatsApp Messenger	2.6.4184	3.70
9	Street View	1.6.0.6	0.28
10	GasBuddy	1.16	1.88

Table 3: Top 10 Apps on Google Android Market. On June 10, 2011, total size 25.5MB, avg size 2.5MB

To better understand why storage affects application performance, we present in Figure 6 a breakdown of the I/O activity during the WebBench workload. Firstly, there are few reads, the majority of which are issued in the beginning of the experiment. There are significantly more writes, with about 2 times more data being written sequentially than randomly. Since the difference between sequential and random performance is greater than a factor of 2 for all SD cards (see Table 2 discussed previously), the time to complete the random writes dominates. Although not shown in the graph, the `/data` partition receives most of the I/O, with only a few reads going to the `/system` partition.

Figure 5 shows the breakdown of CPU utilization for the two workloads; the stacked bar chart shows the CPU tick counts during active, idle, and IOwait periods (a "tick" corresponds to 10ms on our phone). Since the non-idle, non-IOwait CPU consumption includes not only the contribution of the benchmark but also that of all background activities, we also measured how much CPU is consumed in the absence of running the benchmark (for the same amount of time). We found that the share of CPU consumption due to background tasks is less than 1% of the total.

The graph reveals the interesting phenomenon that aggregate CPU consumed for the same benchmark increases with a slower storage device (by just looking at the "active" component). This points to the fact that storage has an indirect impact on energy consumption. We speculate this to be caused either by problems in the software stack, or due to a poorly written application.

4. RELATED WORK

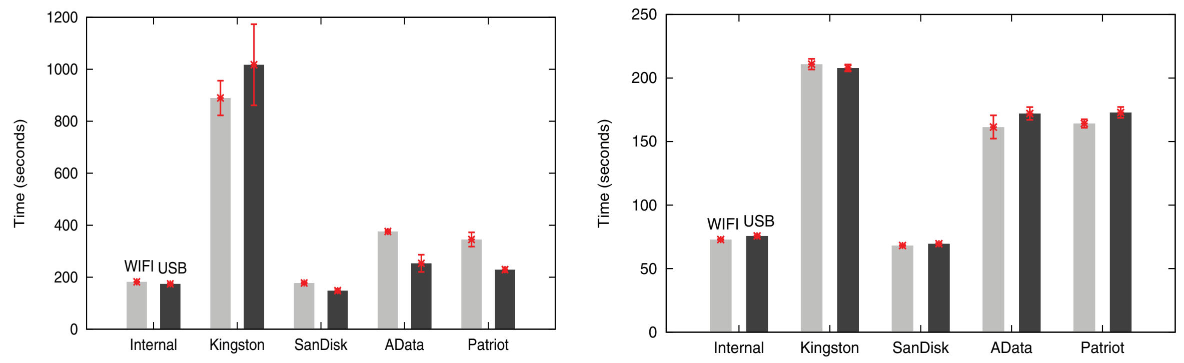
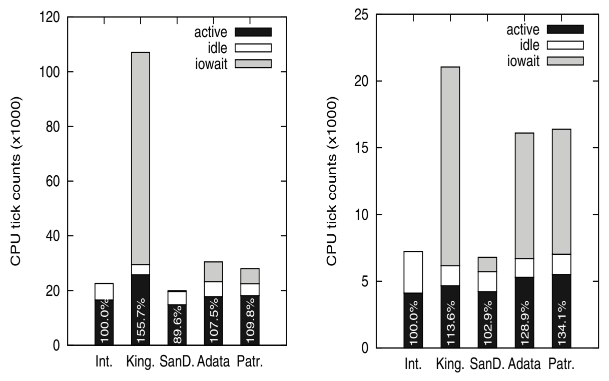


Figure 4: Runtimes for WebBench (left) and AppInstall (right) on Google Nexus One. (a) Shows the total runtime for the WebBench benchmark across the SD cards; each bar represents an average over three trials, and vertical bars represent standard deviation; lighter bar is over WiFi, darker one for USB RT. (b) Similar experiment for AppInstall



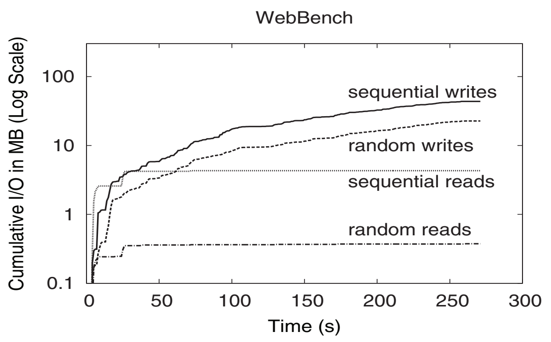


Figure 5: Aggregate CPU used on Nexus One for WebBench (left) and AppInstall (right). Stacked bar shows CPU active, idle, and iowait times; CPU iowait is correlated with runtimes (Fig 4). Even the active times vary across devices – some devices burn more CPU for same amount of work!

Figure 6: WebBench sequential and random I/O activity on Android. Breakdown of I/O issued by WebBench; y-axis is the cumulative I/O issued for the operation type on a log scale, x-axis is time

We found little published literature on storage performance issues for mobile devices. Datalight [8], a provider of data management technologies for mobile and embedded devices to OEMs, makes a similar observations as ours with reference to their proprietary Reliance Nitro file system. According to their website, lack of device performance and responsiveness is one of the important shortcomings of the [Windows] Mobile platforms; OEMs using an optimized software stack can improve performance.

A recent study of web browsers on smartphones [29] examined the reasons behind slow web browsing performance and found that optimizations centering around compute-intensive operations provide only marginal improvements; instead "resource loading" (*e.g.*, files of various types being fetched from the webserver) contributes most to browser delay. While their work focuses more specifically on the browser application and the network, it reaffirms the observation that improvement in OS services and hardware are needed to improve application performance.

Other related work has focused on the implications of network performance on smartphone applications [23], and on the diversity of smartphone usage [20]; power modeling is another area of interest [26]. Keypad [22] and TaintDroid [19] address issues with the privacy and security of data stored on mobile devices. Keypad is a new file system that provides post-theft remote control and fine-grained access auditing. TaintDroid is a real-time privacy monitoring system for smartphones; by integrating taint tracking in the Android platform all the way from the Dalvik VM to the secondary storage, it monitors application behavior to determine when privacy sensitive information is leaked.

Finally, there is extensive work in developing smarter, richer, and more powerful applications for mobile devices, far too much to cite here. We believe the needs of these applications are in turn going to drive the performance requirements expected of hardware devices, including storage devices, as well as their software stacks.

5. CONCLUSIONS

Contrary to conventional wisdom, we find evidence that storage is a significant contributor to the performance of applications on mobile devices; our initial experiments provide some interesting insight into the Android storage stack and reveal the effects of storage on application performance. Surprisingly, we find that even for an "interactive" application such as web browsing, storage can affect the performance in non-trivial ways; for I/O intensive applications, the effects can get much more pronounced. With the advent of faster networks and I/O interconnects on the one hand, and a more diverse, powerful set of mobile applications on the other, the performance requirements expected from storage are only going to increase in the future. We believe the storage subsystem on mobile devices needs a fresh look, and we have taken the first steps in this direction. In the near future, we plan to continue our measurement experiments to better understand the Android operating system and more particularly its storage subsystem. We also plan to further assess the impact of upcoming storage technologies such as storage class memory, on mobile handhelds.

6. REFERENCES

[1] Android Debug Bridge (ADB). `http://developer.android.com/guide/developing/tools/adb.html`.

[2] Block I/O Layer Tracing: blktrace. `http://linux.die.net/man/8/blktrace`.

[3] Busybox unix utilities. `http://www.busybox.net/about.html`.

[4] Clockworkmod rom manager and recovery image. `http://www.koushikdutta.com/2010/02/clockwork-recovery-image.html`.

[5] Compete ranking of top 50 web sites for february 2011 reveals familiar dip. `http://tinyurl.com/3ubxzbl`.

[6] CrystalDiskMark Benchmark V3.0.1. `http://crystalmark.info/software/CrystalDiskMark/index-e.html`.

[7] Cyanogenmod. `http://wiki.cyanogenmod.com/index.php?title=What_is_CyanogenMod`.

[8] Datalight: Software for risk-free mobile data. `http://www.datalight.com/solutions/linux-flash-file-system/performance-hardware-managed-media`.

[9] Google nexus one. `http://en.wikipedia.org/wiki/Nexus_One`.

[10] Htc desire. `http://www.htc.com/www/product/desire/specification.html`.

[11] Starburst data2sd. `http://starburst.droidzone.in/`.

[12] Unrevoked 3: Set your phone free. `http://unrevoked.com/recovery/`.

[13] Usb reverse tethering setup for android 2.2. `http://blog.mycila.com/2010/06/reverse-usb-tethering-with-android-22.html`.

[14] Using databases in android: Sqlite. `http://developer.android.com/guide/topics/data/data-storage.html#db`.

[15] J. Bickford, H. A. Lagar-Cavilla, A. Varshavsky, V. Ganapathy, and L. Iftode. Security versus energy tradeoffs in host-based mobile malware detection. In *MobiSys'11: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, page TBD, Bethesda, Maryland, USA, June/July 2011. ACM Press, New York, NY, USA.

[16] 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*, USENIX ATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[18] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[19] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[20] 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*, MobiSys '10, pages 179–194, New York, NY, USA, 2010. ACM.

[21] Gartner. Gartner highlights key predictions for it organizations and users in 2010 and beyond. `http://www.gartner.com/it/page.jsp?id=1278413`.

[22] R. Geambasu, J. P. John, S. D. Gribble, T. Kohno, and H. M. Levy. Keypad: an auditing file system for theft-prone devices. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 1–16, New York, NY, USA, 2011. ACM.

[23] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 165–178, New York, NY, USA, 2010. ACM.

[24] iperf network performance tool. http://sourceforge.net/projects/iperf.

[25] P. Meroni, E. Pagani, G. P. Rossi, and L. Valerio. An opportunistic platform for android-based mobile devices. In *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*, MobiOpp '10, pages 191–193, New York, NY, USA, 2010. ACM.

[26] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.

[27] Richard Pentin (Summary). Gartner's mobile predictions. `http://ifonlyblog.wordpress.com/2010/01/14/gartners-mobile-predictions/`.

[28] Samsung Corp. Samsung ships industryâĂŹs first multi-chip package with a pram chip for handsets. `http://tinyurl.com/4y9bsds`.

[29] Zhen Wang and Felix Xiaozhu Lin and Lin Zhong. Why are Web Browsers Slow on Smartphones? In *ACM HotMobile '11*, March 2011.