Storage on Your Smartphone Uses More Energy Than You Think

Jayashree Mohan[1] Dhathri Purohith[1] Mathew Halpern[2]
Vijay Chidambaram[1] Vijay Janapa Reddi[2]

[1]Department of Computer Science, University of Texas at Austin
[2]Department of Electrical and Computer Engineering, University of Texas at Austin

Abstract

Energy consumption is a key concern for mobile devices. Prior research has focused on the screen and the network as the major sources of energy consumption. Through carefully designed measurement-based experiments, we show that for certain storage-intensive workloads, the storage subsystem on an Android smartphone consumes a significant amount of energy (36%), on par with screen energy consumption. We analyze the energy consumption of different storage primitives, such as sequential and random writes, on two popular mobile file systems, ext4 and F2FS. In addition, since most Android applications use SQLite for storage, we analyze the energy consumption of different SQLite operations. We present several interesting results from our analysis: for example, random writes consume *15*× higher energy than sequential writes, and that F2FS consumes *half* the energy as ext4 for most workloads. We believe our results contribute useful design guidelines for the developers of energy-efficient mobile file systems.

1 Introduction

The usage of smartphones has seen a remarkable growth in recent times. Smartphones have become the most preferred computing device for a wide variety of applications. Over 70% of the world's population is predicted to use smartphones by 2020 [1]. There will be over a billion Android devices in use by 2020 [2].

Despite the massive outgrowth of mobile devices, limited battery capacity is a major concern for the majority of smartphone users. Improvements in battery technology has been slower than other areas like memory and processor performance [3, 4]. Compared to Moore's law that doubles transistor density every two years, smartphone battery density doubles once every 10 years [5]. Hence, it is important to identify methods to optimize energy consumption in smartphones.

The problem of energy optimization in smartphones has been analyzed from the perspective of the network [6], the CPU [7], and the GPU [8]. However, the contribution of the storage subsystem to energy consumption has not been widely studied and has gained researchers' attention quite recently [9, 10]. The storage subsystem was considered to be insignificant when it comes to the consumption of energy by smartphones. Contrary to established wisdom [11–14], we show through our experiments that the storage stack contributes to about **36%** of the overall energy consumption for workloads dominated by random IO (§2). A random IO workload is quite common, as most applications use SQLite for storage, and multiple applications committing SQLite records at the same time results in random IO at the storage level [15–17]. Therefore, the storage subsystem is a significant contributor to energy consumption for many workloads, and understanding how it consumes energy is important.

In this work, we seek to experimentally analyze and understand how the energy consumption of the storage subsystem varies by IO pattern, file system, and application. Unlike prior work which used specialized hardware [12], we use differential analysis to study energy consumption in commercial mobile phones. We present a comprehensive study on the energy consumption of different file IO operations (§3.1), various SQLite operations (§3.2), and two commonly used applications (§3.3) in two different file systems supported by Android – the default ext4 [18] and the log structured F2FS [19].

Our analysis reveals several interesting results that will be of interest both to the developers of energy-efficient mobile file systems and Android developers:

- Random writes consume **19**× more energy than sequential writes on ext4; on F2FS the ratio is **12**×
- Random reads consume **7**× more energy than sequential reads on ext4; on F2FS the ratio is **8**×
- The most energy-consuming operation is the random write and the least energy-consuming operation is the sequential read. Their energy consumption differs by **32**× in ext4 and **18**× in F2FS.
- Overall, for real-time Android applications F2FS consumes **2**× less energy as compared to Ext4.

Finally, we discuss the implications of our results for the design of energy-efficient file systems for Android devices (§4). Although F2FS does reduce energy consumption compared to ext4, we believe there is significant room for improvement (potentially reducing energy consumption by half).

Component	*Energy Used (J)*	*% of total*
Screen	38.26	37.0%
CPU & Memory	0.64	0.6%
Network	25.40	24.5%
Storage	37.75	36.5%
Total	103.4	100.0%

Table 1: **Energy consumption of each component**. *The table shows that for an IO intensive workload (100 MB of random 4K writes over the network), storage consumes as much energy as the screen, and more than the network.*

2 Motivation: Energy Consumption for IO-Intensive Workloads

To motivate our research, we empirically measure the energy consumed by the storage subsystem and other components for IO-intensive workloads. The challenge is that most hardware power monitoring tools only provide the total energy consumed by the device. Obtaining a breakdown of the energy consumed by different components requires using experimental phones that has specialized hardware support [12]. To obtain this breakdown with a commercial smartphone, we use *differential* analysis: we designed a series of experiments that allow us to tease apart each component's energy consumption.

Experimental Setup. All experiments are performed on a Samsung Galaxy Nexus S phone with a dual-core 1.2GHz Cortex-A9 processor; 32GB internal memory and 1GB RAM running Android 6.0.1 (cyanogenmod 13.0) on Linux 3.0.101 (F2FS enabled) kernel. There is no external SD card and all the IO happens on the internal eMMC flash storage (`/data` partition) of Android. To measure energy, we use a hardware based power measurement tool known as the Monsoon Power Monitor [20], which gives a fine grained energy measurement. We have instrumented the battery of the Android phone to connect to the Power Monitor hardware. All experiments are conducted in airplane mode and at the same level of brightness, averaged over 10 trials.

Differential Analysis. First, we measure the idle state energy consumption (*Screen* in Table 1) by keeping the display on and ensuring that no other application is running in the background. For our analysis, we make the simplifying assumption that the mobile device is in the same energy state throughout (*i.e.,* the device does not go into a low power mode).

After determining the idle state energy, we then determine the energy consumed by the CPU and memory by writing 100 MB to storage in random writes of 4 KB each to an in-memory file system. Since there is no storage or network component to this workload, and we know the idle state energy consumption, we can determine the energy consumption due to CPU and memory operations. We find that the CPU and memory operations consume very little energy for our workload.

To estimate network energy consumption, we re-run the above experiment with the only difference being the writes are sent continuously from a remote server to the mobile device, where it is performed on an in-memory file system. The difference between the energy consumed by the previous experiment and this experiments provides us the energy consumption of the network.

Finally, to estimate the storage energy cost, we run the same IO-intensive workload *locally* by doing the writes on the internal eMMC flash storage and measure the total energy consumed. While doing so, the device is kept in the airplane mode to ensure that there is no network interference. We calculate the energy consumption of storage by subtracting the energy consumed by the screen, CPU, and memory obtained from the previous experiment.

We measure the total energy consumed by doing the IO-intensive workload on internal flash storage, with the writes received over the network from a remote server. This experiment exercises the screen, the network, the storage, and the CPU and memory.

Table 1 shows the results: the storage stack consumes **36%** of the total energy. For an IO-intensive workload, the storage-stack energy consumption is almost equivalent to the energy consumed by the display.

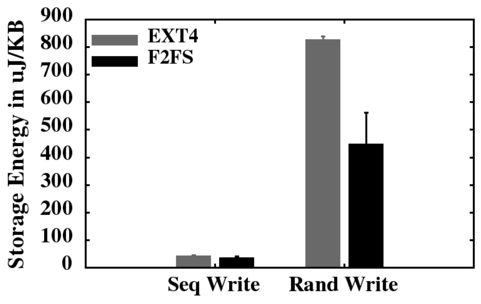
3 Energy Analysis

We now describe our experiments and analysis of energy consumption by different file operations, SQLite operations, and applications in Android.

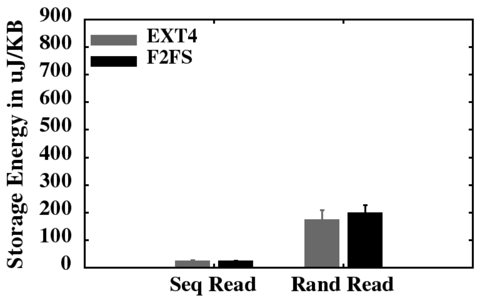
3.1 File IO Operations

We wrote C code to perform sequential and random IO operations and cross-compiled them using `arm-linux-androideabi-gcc` to run on Android. The sequential workload comprises of 1GB of file reads and writes (of IO size 512KB). The random workload includes 100MB of reads and writes (of IO size 4KB) to randomly chosen 4K aligned locations over a 1GB file. Each of these 4K writes are followed by a `fsync()`, to ensure that the writes are persistent on storage. These experiments were performed on two different file systems – ext4 [18] and F2FS [19].

As shown in Figure 1a, random writes consume **19**× more energy than sequential writes in ext4 (**12**× in F2FS). Interestingly, we see that random reads consume 7–8× more energy than sequential reads in both file systems. Comparing across file systems, random writes consume 46% less energy in F2FS compared to ext4, since



(a) Energy Consumption for File Writes



(b) Energy Consumption for File Reads

Figure 1: **Energy Consumption for File IO**. *This graph shows that the energy consumed per KB of sequential write is about 12× – 19× less than that of a random write in F2FS and ext4 respectively. Similarly, the energy consumed per KB of sequential read is about 7× – 8× less than that of a random read in F2FS and Ext4.*

F2FS is a log-structured file system that optimizes random writes. Note that prior work [9] determined the energy consumption of ext4 random writes to be 4000 uJ/KB for cold writes and 300 uJ/KB for warm writes; our workload has a mix of both cold and warm writes, resulting in an average of 800 uJ/KB.

Our workload incurs a lot of write amplification (ratio of total write IO to user data): each random write is followed by an `fsync()` call, and that forces the file system to write both the data and associated metadata (*e.g.,* in ext4, the journal transaction begin/end, list of blocks in transaction *etc.*) to storage. For 10MB of random writes issued, `blktrace` [21] reveals that ext4 actually writes around 70MB of data, whereas F2FS issues only 30MB of writes at the block level. This clearly demonstrates that F2FS reduces write amplification, and hence the energy associated with random writes.

Read performance follows a similar pattern as writes. Random reads consume 7× more energy as compared to sequential reads in ext4 (8× in F2FS). Interestingly, while F2FS consumes less energy for random writes than ext4, F2FS random reads consume 20% *more* energy as shown in Figure 1b. The overhead for F2FS random reads is partially due to NAT table translations. The NAT table holds node IDs to block address mapping, which has to be accessed for each read to fetch the block addresses. For 100MB of random reads, block traces show that ext4 actually reads 195MB, while F2FS reads 272MB at the block level; part of the extra F2FS IO is due to the NAT table.

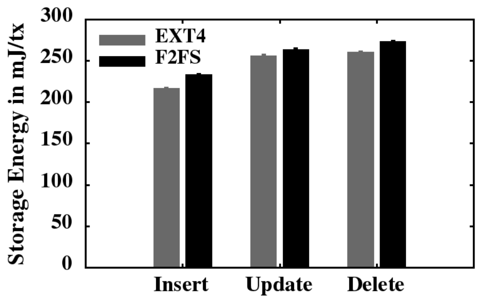
3.2 SQLite Operations

Most Android applications use SQLite as the default database to store app-related information [15,22,23]. We analyze the energy consumed by SQLite operations – Insert, Update and Delete. We have developed an android application to generate different workloads to the SQLite database. The database is pre-populated with a million entries before the start of experiment. If we did not pre-populate a sufficiently large database[1], the random writes would be buffered and written out as sequential updates. We perform 15000 each of SQLite operations (inserts, updates and deletes) for a record size of 4KB. The default SQLite journal mode in Android is set to `DELETE` and the synchronization mode is set to `FULL`. But the most performance efficient mode of SQLite is the Write Ahead Log (`WAL`) journal mode with `NORMAL` synchronization. With `WAL-NORMAL` setting, a SQLite checkpoint is issued after every 1000 pages are written into the database. We report the results obtained by performing the above experiments for both default (`DELETE-FULL`) and `WAL-NORMAL` SQLite modes.

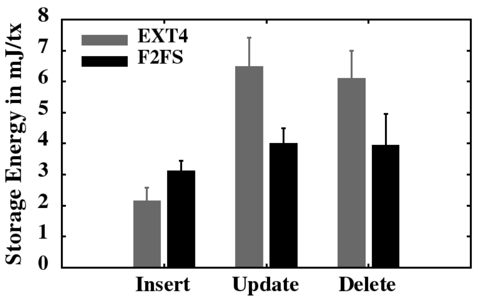
Figure 2a shows the energy consumed by SQLite operations for a transaction of record size 4KB in `DELETE-FULL` mode, which is the default mode in Android. It can be seen that the energy consumed by ext4 is slightly lesser than F2FS for all the SQLite operations. We suspect this is because SQLite issues an `fsync()` after every transaction in `DELETE-FULL` mode, thus nullifying the benefits of sequentialization in F2FS.

Figure 2b shows the energy consumed by SQLite operations for a transaction of record size 4KB in `WAL-NORMAL` mode. As seen in Section §3.1, we expect the energy consumed by F2FS for SQLite operations to be lower than ext4 in this mode. Surprisingly, we see that SQLite inserts consume slightly more energy in F2FS. However, updates and deletes behave as expected and F2FS consumes 1.5× lesser energy than ext4, since F2FS sequentializes the IO. Since inserts are sequential appends while updates and deletes are random, inserts consume equal or lower energy than updates and deletes in both file systems.

[1]We configured the database to be 10× larger than each operation



(a) Energy consumption for `DELETE-FULL` SQLite mode



(b) Energy consumption for `WAL-NORMAL` SQLite mode

Figure 2: **Energy Consumption for SQLite operations**. *This graph shows that the energy consumed per transaction for SQLite operations are slightly higher in F2FS as compared to ext4 in* `DELETE-FULL` *SQLite mode. However, in* `WAL-NORMAL` *SQLite mode, the energy consumed per transaction for SQLite updates and deletes are about 1.5× lower, while inserts consume about 1.3× higher energy in F2FS as compared to ext4.*

3.3 Applications

We study two popular, storage-intensive applications on Android devices, Mail and Facebook [24–26]. We enable tracing in the kernel and capture block-level IO traces using `blktrace` [21]. We study the application IO to determine its IO pattern and thereby estimate energy usage.

Workload. For the Mail app, we capture the IO traces over a period of 3 minutes, which is fairly representative of real-world interactivity, during which we first log in to the email account, sync a total of 100 mails, and logout [9]. For the Facebook app, over a duration of 180 seconds, we capture the following activity traces – logging in, loading the news feed, updating the profile picture and sharing a couple of status messages [15]. To ensure that no other application does IO over the trace duration, we kill all the background applications and have no other extra application installed from the play store.

Computing Randomness. We make use of the `blktrace` version that has the ability to trace flush requests by tracking the `REQ_FLUSH` flag in the bio request. We merge the IO requests that are sequential between the two flush requests in the blktrace. After merging all such successive requests, if the size of the IO is less than 32 KB, it is tagged as random IO. By tagging all requests as sequential or random in this manner, we can compute what percentage of requests results in random IO.

Figure 3 shows the percentage of randomness as seen by the underlying flash device for the two file systems. For the Mail application, both the file systems write out around 110–120MB of data in total. For the Facebook app, both file systems write about 30–35MB of data in total. From the block traces, we see that F2FS issues larger IOs to the storage device than ext4.

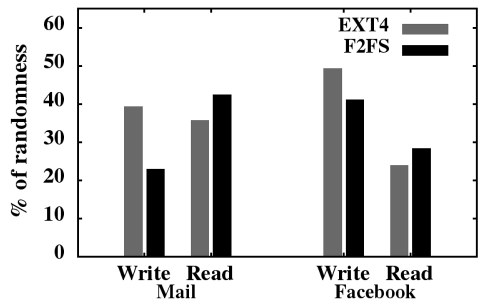


Figure 3: **Percentage random I/O at block level**. *This graph shows that the percentage of random writes in F2FS is* 1.5×–2× *lesser than ext4, where as F2FS does more random reads compared to ext4.*

Estimating Energy Consumption. We estimate the energy consumed by these applications based on our results from (§3.1). Once we have an estimate of the percentage of random and sequential IO, and the total amount of data read and written, we can estimate the total energy consumed by the application and the average IO size.

For the given workload, the estimated energy consumption for Mail is approximately 42.91J in ext4 and 20.07J in F2FS. Similarly, for the Facebook app, the energy estimation is approximately 14.13J in ext4, and 8.79J for F2FS. Thus, F2FS reduces energy consumption by 2× when both applications are considered.

3.4 Limitations

Our study only examines the energy consumption of the underlying file-system and SQLite operations; the actual energy consumed by applications depends on their usage of the storage stack. While we do study two popular applications, our results are not representative of all applications or of other workloads for these applications. We plan on expanding our study in the future.

4 Implications for File-system Design

Use Sequential IO. Since the energy consumed by sequential I/O is almost a magnitude less than that consumed by random I/O, an energy-efficient file system should replace random I/O by sequential I/O as much as possible. F2FS attempts this with its log-structured design, achieving an impressive 2× reduction in energy consumption when compared to ext4. Although this improvement is significant, F2FS still performs around 20–28% of random writes and about 12–20% of random reads. Since random writes are so expensive in terms of energy, sequentializing the last 20–28% of random writes can reduce energy consumption by *half*.

Seq Write vs. Rand Read Trade Off. Our experiments show that while file systems can save energy by writing data sequentially (*e.g.,* using a log-structured approach), it can result in more random reads. Random reads are 7–8× more expensive than sequential reads, so it may not be a good fit for applications that do a lot of random reads. Thus, it is not as simple as just make all writes sequential; there is a trade-off involved based on application IO patterns.

Compress IO. We found that for IO-intensive workloads, the CPU energy consumption is low. This indicates that the file system should compress the data (by using one of the idle cores [27]) before performing IO.

5 Related Work

Although prior research has investigated the energy consumption of the network [6,13], the CPU [7,27], and the GPU [8], recent research from Microsoft Research and UC San Diego was the first to measure the energy consumption of storage [9]. Li *et al.* showed that the storage stack consumes 200× the energy of storage hardware, pointing out that upgrading storage hardware alone will not reduce storage energy consumption. Our independent study replicates some of their results on different hardware, and goes further to analyze the cost of different file and SQLite operations.

Our work also analyzes the randomness of application IO and ties it to the application energy cost. Similar analyses have been done by prior work, which pointed out that the random IO of applications is a big factor in their poor performance on mobiles [15].

While prior work has measured component-wise power consumption on an experimental phone with specialized hardware support [12], we are the first to do so on a commercially available Android device.

Pathak *et al.* present an energy profiler for Android apps which identified that most of the energy in an app is spent in accessing the I/O components [28]. However their work does not provide a fine grained energy analysis of the storage subsystem, as we do. Nguyen *et al.* have analyzed how different storage stack configurations (*e.g.,* different IO schedulers) impacted performance [10]. While their work analyzed the impact of storage-stack parameters on storage energy consumption, we believe our work is the first to show that storage can consume as much energy as the network or the display for IO-intensive workloads.

There have been other attempts to reduce energy consumption on mobile devices. MobiFS trades durability for improved energy efficiency in smartphones [29]. Based on our results, we believe we can achieve greater reduction in energy consumption than MobiFS without sacrificing durability. Flashlogger uses amnesic compression techniques to lower energy costs [30]. Although FlashLogger is designed for sensor systems, its optimizations are applicable to mobile systems with limited memory and processing power.

6 Conclusion

Prior work in measuring energy consumption has used specialized hardware; we use differential analysis to measure energy consumption of different components on a commercial mobile phone. Traditionally, the screen and the network have been considered to consume the most energy on a mobile device; we present experimental evidence that for IO-intensive workloads, storage can consume more energy than the network, and as much energy as the display. We analyze the energy consumption of different file operations and show that random IO consumes significantly more energy than sequential IO. We find that F2FS reduces the energy consumption of most SQLite operations (compared to default ext4). We hope our work spurs further research in energy-efficient file systems; we believe file systems such as F2FS can be made significantly more energy-efficient.

Acknowledgments

We would like to thank the anonymous reviewers for their feedback and guidance. This work was supported in part by NSF (CCF-26100887), and by generous donations from VMware and Facebook. Any opinions, findings, and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of other institutions.

References

[1] Ericson report. `https://www.ericsson.com/news/1925`, December 2016.

[2] IDC report. `http://www.idc.com/getdoc.jsp?containerId=prUS41425416`, December 2016.

[3] Roy Want. The power of smartphones. *IEEE Pervasive Computing*, 13-03:76–79, 2014.

[4] Anirudh Badam, Ranveer Chandra, Jon Dutra, Anthony Ferrese, Steve Hodges, Pan Hu, Julia Meinershagen, Thomas Moscibroda, Bodhi Priyantha, and Evangelia Skiani. Software defined batteries. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 215–229. ACM, 2015.

[5] Fred Schlachter. No moores law for batteries. *Proceedings of the National Academy of Sciences*, 110(14):5273–5273, 2013.

[6] Shuo Deng and Hari Balakrishnan. Traffic-aware techniques to reduce 3g/lte wireless energy consumption. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 181–192. ACM, 2012.

[7] Antti P Miettinen and Jukka K Nurminen. Energy efficiency of mobile clients in cloud computing. *HotCloud*, 10:4–4, 2010.

[8] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[9] Jing Li, Anirudh Badam, Ranveer Chandra, Steven Swanson, Bruce Worthington, and Qi Zhang. On the energy overhead of mobile storage systems. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, FAST'14, pages 105–118, Berkeley, CA, USA, 2014. USENIX Association.

[10] David T. Nguyen, Gang Zhou, Xin Qi, Ge Peng, Jianing Zhao, Tommy Nguyen, and Duy Le. Storage-aware smartphone energy savings. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 677–686, New York, NY, USA, 2013. ACM.

[11] Ding Li, Shuai Hao, Jiaping Gui, and William GJ Halfond. An empirical study of the energy consumption of android applications. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 121–130. IEEE, 2014.

[12] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14. Boston, MA, 2010.

[13] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[14] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 168–178. IEEE, 2009.

[15] Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. Revisiting storage for smartphones. *Trans. Storage*, 8(4):14:1–14:25, December 2012.

[16] Kisung Lee and Youjip Won. Smart layers and dumb result: Io characterization of an android-based smartphone. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 23–32. ACM, 2012.

[17] Sooman Jeong, Kisung Lee, Seongjin Lee, Seoungbum Son, and Youjip Won. I/o stack optimization for smartphones. In *USENIX Annual Technical Conference*, pages 309–320, 2013.

[18] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33. Citeseer, 2007.

[19] Changman Lee, Dongho Sim, Joo-Young Hwang, and Sangyeun Cho. F2fs: A new file system for flash storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 273–286, Berkeley, CA, USA, 2015. USENIX Association.

[20] Monsoon Power Monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`, December 2016.

[21] Block I/O Layer Tracing. `https://linux.die.net/man/8/blktrace`, December 2016.

[22] SQLite for Android. `https://developer.android.com/guide/topics/data/data-storage.html#db`, December 2016.

[23] Every Android Phone uses SQLite. `https://www.sqlite.org/mostdeployed.html`, December 2016.

[24] Popular Mobile Apps. `https://qz.com/481245/these-are-the-25-most-popular-2015-mobile-apps-in-america/`, May 2017.

[25] Popular Mobile Apps. `http://time.com/4156902/most-popular-apps-2015/`, May 2017.

[26] Storage Intensive Mobile Apps. `http://www.indiatimes.com/technology/apps/65-android-apps-that-are-killing-your-battery-data-and-storage-243825.html/`, May 2017.

[27] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pages 64–76. IEEE, 2016.

[28] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 29–42. ACM, 2012.

[29] Jinglei Ren, Chieh-Jan Mike Liang, Yongwei Wu, and Thomas Moscibroda. Memory-centric data storage for mobile systems. In *USENIX Annual Technical Conference*, pages 599–611, 2015.

[30] Suman Nath. Energy efficient sensor data logging with amnesic flash storage. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 157–168. IEEE Computer Society, 2009.