Smartphone Application Launch with Smarter Scheduling

David T. Nguyen
College of William and Mary
McGlothlin-Street Hall 126
Williamsburg, VA 23185, USA
dnguyen@cs.wm.edu
Ge Peng
College of William and Mary
McGlothlin-Street Hall 126
Williamsburg, VA 23185, USA
gpeng@cs.wm.edu
Daniel Graham
College of William and Mary
McGlothlin-Street Hall 126
Williamsburg, VA 23185, USA
dggraham@cs.wm.edu
Gang Zhou
College of William and Mary
McGlothlin-Street Hall 126
Williamsburg, VA 23185, USA
gzhou@cs.wm.edu

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s).
UbiComp'14 Adjunct, September 13-17, 2014, Seattle, WA, USA
ACM 978-1-4503-3047-3/14/09.
http://dx.doi.org/10.1145/2638728.2638763

Abstract
The time it takes to launch a smartphone application is unpredictable. In this paper, we explore how these unpredictable launch times are affected by constraints associated with reading (writing) from (to) flash storage. We conduct the first large-scale measurement study on the Android I/O delay using the data collected from our Android application running on 1480 devices within 188 days. Among others, we observe that reads experience up to 626% slowdown when blocked by concurrent writes. We use this obtained knowledge to design a pilot solution, wherein by prioritizing reads over writes we are able to reduce the launch delay by up to 37.8%.

Author Keywords
Smartphone Responsiveness; Application Delay; I/O Optimizations; Application Launch

ACM Classification Keywords
C.4 [Performance of Systems]: Design studies.

Introduction
A recent analysis [11] indicates that most user interactions with smartphones are short. Specifically, 80% of the apps are used for less than two minutes. With such brief interactions, apps should be rapid and responsive. However, the same study reports that many apps incur significant delays during launch and run-time. This work addresses two key research questions towards achieving

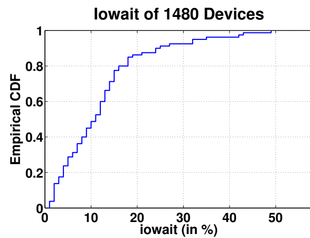

Figure 1: Iowait.

Figure 2: StoreBench Android App.

rapid app response. (1) *How does disk I/O performance affect smartphone app response time?* (2) *How can we improve app performance with I/O optimization techniques?*

In order to understand how disk I/O performance affects smartphone application response time, we conduct a series of measurement studies. First, we investigate what portion of the CPU active time Android devices spend in storage waiting for I/Os to complete. When the time the CPUs spend in the storage subsystem is significant, this will negatively affect the smartphone's overall application performance, and result in slow response time. To identify what may be causing such waits, we learn more about I/O activities and their properties. The property that may be a reason of such waits is I/O slowdown, which quantifies how one I/O type is slowed down due to presence of another. If one I/O activity (e.g., read) is slowed down by another (e.g., write), there will be certain cases in the application life cycle that will suffer from such slowdown (e.g., launch, since reads dominate during launch).

We summarize our contributions as follows:

- First, through a large-scale measurement study based on the data collected from 1480 devices using an app we developed, we find that Android devices spend a significant portion of their CPU active time (up to 58%) waiting for storage I/Os to complete. Further investigation reveals that a read experiences up to 626% slowdown when blocked by a concurrent write.
- Second, we design and implement a system prototype called SmartIO that shortens the application delay by prioritizing reads over writes.
- Third, we evaluate our system using 20 popular applications from four groups (sensing, regular, streaming, and games) and we show that SmartIO reduces launch delays by up to 37.8%, and run-time delays by up to 29.6%.

Motivation

In order to understand how disk I/O performance affects smartphone application response time, we conduct a large-scale measurement study using the data collected from our Android app running on 1480 Android devices. The app is available for free download at [3], and is displayed in Figure 2. The results in Figure 1 reveal that Android devices spend a significant portion of their CPU active time waiting for storage I/Os to complete, also known as *iowait*. Specifically, around 40% of the devices have *iowait* values between 13% and 58%. This negatively affects the smartphone's overall application performance, and results in slow response time. Therefore, in order to improve the application performance, it is essential to investigate possible causes of such waits.

Further investigation identifies that one of the reasons causing such waits is I/O slowdown, which represents the slowdown of one I/O type due to presence of another. In particular, our experimentation reveals a significant slowdown of reads in the presence of writes. Specifically, a sequential read experiences up to 626% slowdown when blocked by a concurrent write. Similarly, a random read experiences up to 293% slowdown when blocked by a concurrent write. This significant read slowdown may negatively impact the application performance during the life cycles when the number of reads dominates. A good example is application launch.

Pilot Solution

In order to improve the application delay performance in smartphones, we present our pilot solution called SmartIO [9, 8, 6, 5], a system that reduces the application response time by prioritizing reads over writes, and grouping them

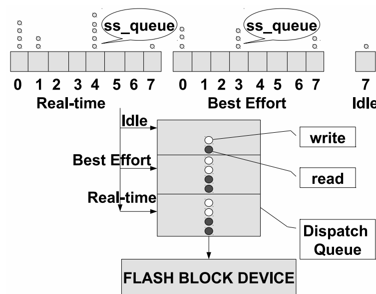

Figure 3: Dispatch Example.

based on assigned priorities. SmartIO issues I/Os with optimized concurrency parameters.

I/O Priority Assignment. Our system follows the implications from the previous measurement study. First, since a read suffers a large slowdown in the presence of a concurrent write, the goal is to allow reads to be completed before writes, while avoiding write starvation. In order to achieve this, a third level of I/O priority is added into the current block layer [1], assigning higher priority to reads and lower to writes. This third priority level has a lower priority than the first two priority levels in the existing Linux I/O scheduler (CFQ). Details will be elaborated in the following paragraph. Write starvation is avoided by applying a maximal period of time assigned to a process, which is by default 100ms as used in the CFQ's time slice concept. Beside CFQ, the proposed solution can be easily adapted to other schedulers.

I/O Dispatch. A sample dispatch is illustrated in Figure 3. In the current CFQ implementation, each block device has 17 queues of I/O requests (8 Real-time, 8 Best Effort, and 1 Idle). The existing system selects a queue based on the priorities, takes a request in the queue, and inserts it in the dispatch queue. The queue selection process accounts for two priority levels: the class priority (Real-time, Best Effort, Idle), and the priority within the class (0-7). Our system does not change the above dispatch process but uses a third priority level to organize the dispatch queue in favor of the reads. The dispatch queue is then divided into three sections, from the bottom up real-time, best effort, and idle requests. Each section is organized such that reads precede writes.

Evaluation

We evaluate our pilot solution by measuring launch and run-time delays of 20 popular apps (5 sensing, 5 regular, 5 streaming, and 5 games), with and without SmartIO. During the experiment, our Samsung S4 phone has all radio communication disabled except for WiFi that is necessary to provide stable Internet connections required on most apps. The screen is set to stay-awake mode with constant brightness, and the screen auto-rotation is disabled. Only one app runs at a time, and no other app is in the background. The cache is cleared before each measurement in order to evaluate real performance improvement caused by SmartIO.

The Android Monkey tool [2] is utilized to trigger the launch process of each app. The application *launch delay* starts when the launch process is triggered, and ends when the process completes. The delays include four components obtained through the Linux *time* command: the time taken by the app in the user mode (*user*), the time taken by the app in the kernel mode (*system*), the time the app spends waiting for the disk I/Os, and the time the app spends waiting for the network I/Os. In order to test delays of apps running on the phone with SmartIO, we utilize again the Android Monkey tool to generate streams of 500 user events such as clicks, touches, or gestures. The *run-time delay* is defined as the time needed to complete the 500 user events in a running app.

The results indicate that SmartIO reduces launch delays by up to 37.8%, and run-time delays by up to 29.6% (Figure 4). SmartIO's performance gain during launch is due to its read-intensive nature. Specifically, the average number of reads observed during launch on the 20 apps is five times higher than writes. The smaller performance gain during the run-time is caused by its modest I/O activity. SmartIO's read performance improvement comes with little cost due to the read/write discrepancy nature of the flash storage (reads take much faster to complete).

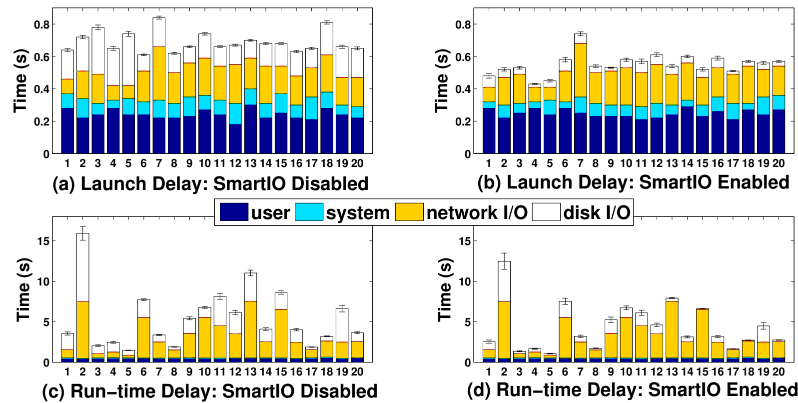

Figure 4: Launch and Run-time Delay. **1:**Angry Birds; **2:**GTA; **3:**NFS; **4:**Temple Run; **5:**The Simpsons; **6:**CNN; **7:**Nightly News; **8:**ABC News; **9:**YouTube; **10:**Pandora; **11:**Facebook; **12:**Twitter; **13:**Gmail; **14:**Maps; **15:**AccuWeather; **16:**Accelerometer; **17:**Gyroscope; **18:**Proximity Sensor; **19:**Compass; **20:**Barometer.

Future Work

Although SmartIO is designed to favor reads over writes, it may be sometimes desirable to have the ability of lowering this read-preference, or even changing it to write-preference for certain workloads. This more accurate priority control will require further understanding of the read and write priority. Finally, our experience from previous work [7, 4] will guide us in balancing the energy and performance trade-off through various I/O optimization techniques.

Related Work

Little work in the research community directly relates to ours. Yan et al. [11] and Parate et al. [10] propose systems predicting application launch to reduce the launch delay. However, mispredictions of the proposed approaches will lead to significant memory and energy overhead.

Conclusion

This paper presents a measurement study on the behavior of reads and writes in smartphones. The obtained insights are used to design and implement a system that reduces the application delay by prioritizing reads over writes.

References

[1] Block layer. http://goo.gl/SwdLZ5, 2014.

[2] Monkey. http://goo.gl/F14hW, 2014.

[3] Storebench download. http://goo.gl/ava9eV, 2014.

[4] Nguyen, D. T. Evaluating impact of storage on smartphone energy efficiency. In *Proc. of ACM UbiComp* (2013).

[5] Nguyen, D. T. Improving smartphone responsiveness through i/o optimizations. In *Proc. of ACM UbiComp* (2014).

[6] Nguyen, D. T. Smartphone application delay optimizations. In *Proc. of ACM MobiSys* (2014).

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

[8] Nguyen, D. T., Zhou, G., and Xing, G. Poster: Towards reducing smartphone application delay through read/write isolation. In *Proc. of ACM MobiSys* (2014).

[9] Nguyen, D. T., Zhou, G., and Xing, G. Video: Study of storage impact on smartphone application delay. In *Proc. of ACM MobiSys* (2014).

[10] Parate, A., Böhmer, M., Chu, D., Ganesan, D., and Marlin, B. M. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proc. of ACM UbiComp* (2013).

[11] Yan, T., Chu, D., Ganesan, D., Kansal, A., and Liu, J. Fast app launching for mobile devices. In *Proc. of ACM MobiSys* (2012).