Smartphone Application Delay Optimizations

David T. Nguyen
College of William and Mary
McGlothlin-Street Hall 126
Williamsburg
VA 23185, USA
dnguyen@cs.wm.edu

ABSTRACT

Despite the rapid hardware upgrades, current smartphones suffer various unpredictable delays during operation, e.g., when launching an app, leading to poor user experience. In this work, we investigate the behavior of reads and writes in smartphones. We conduct the first large-scale measurement study on the I/O delay of Android using the data collected from our Android app running on 1009 devices within 130 days. Among other factors, we observe that reads experience up to 626% slowdown when blocked by concurrent writes for certain workloads. We use this obtained knowledge to design a pilot solution called SmartIO that reduces application delays by prioritizing reads over writes. SmartIO is implemented on the Android platform and evaluated extensively on several groups of popular applications. The results show that our system reduces launch delays by up to 37.8%, and run-time delays by up to 29.6%.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies

Keywords

Smartphone App Delay; I/O Optimizations; App Launch

1. INTRODUCTION

A recent analysis[8] 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 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 plan to conduct a series of measurement studies. First, we want to 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 intend to learn more about I/O activities and their properties. The first 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). Another property to be researched is concurrency. Depending on hardware characteristics, different devices may benefit differently from concurrency. Therefore, we plan to study the speedup of concurrent I/Os over serial ones.

We summarize our thesis statement as follows:

- Investigate the impact of storage I/O performance on smartphone application delay.
- Explain root reasons of such impact.
- Develop storage-aware solutions with fast application response.

If we succeed, we will contribute to better understanding of the limitations of current day and future smartphone devices, i.e., how and why such devices exhibit significantly different application performance when different storage policies are applied in the I/O path and what device and system improvements are necessary. We will also contribute specific innovative storage-aware solutions with fast application response that work well in practice.

Little work in the research community directly relates to ours. Yan et al.[8] and Parate et al.[7] propose systems predicting application launch to reduce the launch delay. Their systems reduce perceived delay through application prelaunching. However, mispredictions of the proposed approaches will lead to significant memory and energy overhead.

2. 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[2] running on 1009 Android devices. The results reveal that Android devices spend a significant portion of their CPU active time waiting for storage I/Os to complete (Figure 1). 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

MobiSys'14 PhD Forum, June 16, 2014, Bretton Woods, NH, USA.

http://dx.doi.org/10.1145/2611166.2611168.

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. Finally, the last property researched is concurrency. Our study suggests that different devices may benefit differently from concurrency. Therefore, we need to be able to adapt to the concurrency characteristics of each device.

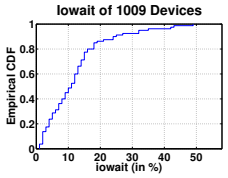

Figure 1: Iowait.

3. PILOT SOLUTION

In order to improve the application delay performance in smartphones, we present our pilot solution called SmartIO[6, 5], a system that reduces the application response time by prioritizing reads over writes, and grouping them 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 2. 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.

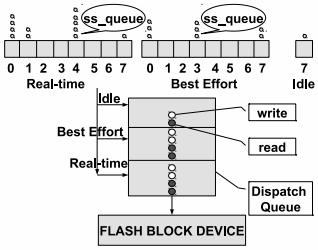

Figure 2: Dispatch Example.

Concurrency Profiler. The system uses the knowledge of the device's four concurrency parameters to issue I/Os from the dispatch queue. The parameters include the optimal number of sequential or random reads (writes) that benefit most from concurrency. To achieve this, SmartIO measures the concurrency parameters during installation by issuing reads and writes, and calculates the speedup of concurrent I/Os over serial ones.

We evaluate our pilot solution using 20 popular apps and show that SmartIO reduces launch delays by up to 37.8%, and run-time delays by up to 29.6% (Figure 3). 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).

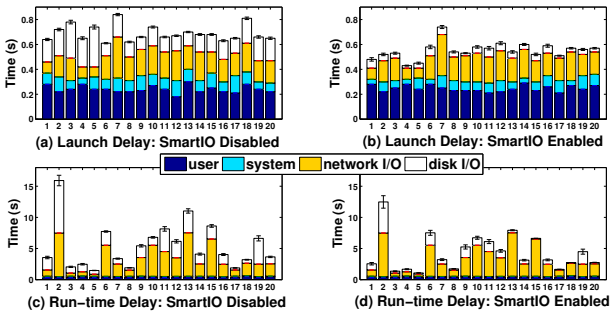

Figure 3: Launch and Run-time Delay. **1:**Angry Birds; **2:**GTA; **3:**NFS; **4:**Temple Run; **5:**The Simpsons; **6:**CNN; **7:**Nightly News; **8:**ABC News; **9:**YouTube; **10:**Pandora; **11:**Facebook; **12:**Twitter; **13:**Gmail; **14:**Maps; **15:**AccuWeather; **16:**Accelerometer; **17:**Gyroscope;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

4. REMAINING STEPS

We plan to conduct more controlled performance evaluations. In particular, we intend to choose a group of I/O intensive and a group of I/O non-intensive apps, and determine the delay performance gain. Additionally, the impact of our system on writes will be studied. It will be important to have deeper understanding of what kinds of workloads are benefiting from SmartIO, and how prevalent they are in apps.

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. We will have to investigate how to determine such priorities, and how to dynamically and efficiently enforce them in the system. In particular, we plan to collect I/O patterns from a large number of apps, and train a supervised classification model or an unsupervised clustering model for run-time priority assignment.

Finally, we expect that the energy overhead will fluctuate as we introduce a goal oriented approach to controlling the read/write priority. Our experience from previous work[4, 3] will guide us in balancing the energy and performance trade-off through various I/O optimization techniques.

5. REFERENCES

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

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

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

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

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

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

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

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