Improving User Experience on Low-end Mobile Devices

Muhammad Abdullah, Muhammad Mahad

I. Introduction

Mobile devices are becoming increasingly dominant in the developing world as a greater selection of devices becomes available. However, a significant fraction of mobile users in the developing world own low-end devices. Modest hardware resources and varying software configurations coupled with the complexity of modern applications have presented new challenges to researchers for empowering entry-level devices. Despite the rapid technological evolutions in the mobile world, system developers and application developers have failed to provide a uniform user experience across devices. Content adaptation and system-level optimizations play a key role in bridging the gap between device capabilities and mobile QoE (Quality of Experience). Improving user experience on low-end devices would help in connecting a large population of users in the developing countries to the world of modern and complex mobile applications.

II. Related Work

Various techniques and research directions have been explored recently for improving the mobile QoE on entry-level mobile devices. Broadly speaking, the researchers have taken two directions in this regard: Content adaptation and system-level optimizations.

1) Content adaptation

While adapting content for the end-users, the key questions researchers try to answer are where the adaptation should take place, when it should be performed, and what content should be adapted. The efforts in this regard have been along three possible directions: i) Server-side optimizations, ii) Client-side optimizations, and iii) Proxy services.

i) Server-side optimizations
Server-side adaptation involves adapting resources and software on the content delivery servers. Facebook has designed it's Free Basics [5] service and its restrictions with assumptions about users' device capabilities (e.g., lack of JavaScript support). It removes Javascript and rich multimedia from the web services, to support the browsers on entry-level mobile devices. The main advantage of server-side customizations is that every web page can be tailored for the specific needs of end-users.

ii) Client-side optimizations
On the other end, we have the client-side optimizations where the content is adapted on the endpoint itself. Zafar et al. [2] investigated mobile user's web browsing experience under memory pressure across a variety of mobile devices with varying hardware capabilities. They found that memory footprint of webpages was much larger than their sizes and under high memory pressure, the average Page Load Time (PLT) of webpages increased sharply. They proposed client-side optimizations for reducing the memory footprint of web pages that include debloating JS by using native JS code, removing unused functions from JS libraries, and choosing memory-efficient image formats. This resulted in significant improvement in PLTs and thus end-user experience. However, the only

drawback of such approaches is that users have to install third-party applications or browser plug-ins for transforming content locally.

iii) Proxy services

Proxy-based adaptation is performed between the content provider and the client. It can be implemented by the content provider or by a third party. Google's Web Light [1] proxy-based transcoding service is a popular example in this regard. It detects users on slow networks and automatically delivers lighter webpages on the fly that are optimized for slow devices, reducing PLT and saving user data. However, Web Light supports only a limited number of Ad networks, which hurts the ad revenue of publishers, and does not transcode all webpages. Moreover, Web Light supports HTTPS pages but does not preserve end-to-end encryption and thus can easily read user data and build users' browsing profiles. Due to privacy concerns, proxy services are not gaining much traction in the research spheres.

2) System-level optimizations

There are several ongoing efforts for improving mobile QoE through system-level optimizations. These include specialized operating systems for low-end devices(e.g, Android Go [6] and Puffin OS [7]). Android currently employs "kswapd" and "lmkd" kernel daemons for reclaiming cached pages and killing background applications respectively. The reclaim mechanism is based on watermarks, keeping a certain percentage of memory free. This is a slow and power-hungry process and results in poor user experience in foreground applications during high memory pressure by consuming significant CPU cycles. [3] proposes proactively reclaiming cached pages before the memory gets tight. However, as they demonstrate, proactively reclaiming all idle cached pages can result in even more application kills and cold starts depending on the app cycle order. In addition, they demonstrate that deactivating cached pages and performing application compaction (through in-memory compression) while the system is not under memory pressure reduces application kills and cold starts. On the other end of the spectrum, Google [4] has also proposed two new kernel threads (kstaled and kreclaimd) for proactive reclaims in its datacenter that hint kernel about processes least likely to be used in the future. kstald scans for idle pages and tags them using the existing page flags, incurring no memory overhead. It then forwards a queue of idle pages to kreclaimd for memory reclaim. They state that they do not use the existing kswapd daemon for proactive reclaims because it is built on a complicated set of heuristics that they do not want to modify.

III. Proposed Approach

The heterogeneous nature of mobile devices makes it desirable to adapt the system-level optimizations, specifically memory management and scheduling, on a per device basis. As a first step, we want to statically learn application memory access patterns across a variety of mobile devices. We aim to analyze memory traces of popular Android applications on a variety of devices. Moving forward, the direction we wish to explore is coming up with an automated mechanism for analyzing memory access patterns. In this regard, we could train a machine learning model based on carefully selected features that learn application behaviors in terms of memory profile and can generate accurate predictions about memory usage. These predictions can help the kernel perform memory management and scheduling in a more informed way by adapting and tailoring the watermarks and thresholds for individual devices.

References

[1]Ammar Tahir, Muhammad Tahir Munir, Shaiq Munir Malik, Zafar Ayyub Qazi, Ihsan Ayyub Qazi. "Deconstructing Google’s Web Light Service."
[2]Zafar Ayyub Qazi, Ihsan Ayyub Qazi, Theophilus Benson, Ghulam Murtaza, Ehsan Latif, Abdul Manan, Abrar Tariq. “Mobile Web Browsing Under Memory Pressure”
[3]www.linuxplumbersconf.org/event/4/contributions/282/attachments/365/598/Finding_more_DRAM.pdf
[4] https://lwn.net/Articles/787611
[5] Sen, R., Ahmad, S., Phokeer, A., Farooq, Z. A., Qazi, I. A., Choffnes, D., & Gummadi, K. P. (2017). Inside the walled garden: Deconstructing facebook's free basics program. ACM SIGCOMM Computer Communication Review, 47(5), 12-24.
[6] https://www.android.com/versions/go-edition/
[7] https://www.puffin.com/os/