Mobifetch: An Execution-time Prefetching Technique to Expedite Application Launch in Mobile Devices

Junhee Ryu, Kwangjin Ko, Heonshik Shin
Seoul National University
{jhryu, kjko, shinhs}@cslab.snu.ac.kr

Kyungtae Kang
Hanyang University
ktkang@hanyang.ac.kr

ABSTRACT

This paper presents a novel prefetching technique to reduce application launch time for mobile devices. The proposed method traces disk access accurately during an application launch and prefetches them in efficient way at its subsequent launches. The key idea is to parallelize the use of processor and flash disk while exploiting multi-core and internal parallelism on flash disk. The proposed prefetcher implemented on a mobile Meego platform has achieved a 28.1% reduction of application launch time with 6 popular applications.

1. INTRODUCTION

Application launch time is an important index to benchmark the user-perceived system performance on mobile devices [1]. Emergence of flash-based disk has shown a great potential to make application launch rather more processor-intensive. We observed, however, that significant portion of application launch time is still related to disk time which includes I/O stack processing, context switch overhead, and low-level disk I/O latency. We also found that hardware resources are used in serialized manner during application launch although the process can benefit by exploiting a parallelism between processors and disk drives [1]. To tackle the inefficient use of system resources, we propose a new prefetching technique called *Mobifetch* to expedite application launch by exploiting multi-core, internal parallelism on flash disk, and concurrent use of the processor and the disk in the Linux environment.

2. MOBIFETCH DESIGN AND IMPLEMTATION

Disk I/O Tracing: The accuracy of disk I/O tracing critically affects the achievable performance gain on prefetching techniques. To figure out exact set of accessed blocks during an application launch, we first invalidate disk cache and monitor generated I/O requests due to disk cache miss in buffer cache and page cache. We also perform filesystem-level block dependency check because Linux kernel does not perfectly invalidate slab caches.

Prefetch Scheduler: Collected launch-related blocks are scheduled to optimize application launch time by utilizing system resources in parallel. We use *infill merge* [2], which merges I/O requests with small unneeded blocks between them, to exploit internal parallelism on flash disk. In addition, we consider the order of requests for the blocks to be merged. This is because we need to avoid the merge of blocks located at a large distance, which is deemed inefficient for overlapping processor execution with disk prefetching. We use I/O request distance and infill size thresholds to limit the distance between mergeable blocks and the size of extra reading blocks.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012R1A1A2044653)

Prefetch Thread: Launch-related blocks of applications are monitored, scheduled, and stored to .pf file at their first launch. Optimized sequence is prefetched by a prefetcher thread using the stored .pf file at subsequent launches of each application.

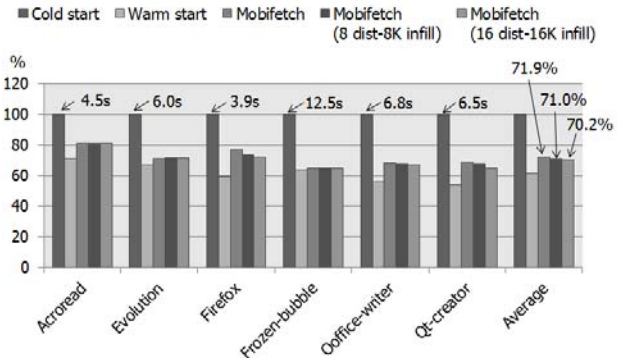

Figure 1. Results of application launch time measurement (normalized to cold start time).

3. EXPERIMENTAL RESULTS

We conducted an experiment on a tablet PC equipped with an Intel dual-core atom 1.86 GHz CPU and a Sandisk 64 GB SSD (SDSA3ED, this model does not support command queuing), where we installed a Meego 1.2 with Mobifetch-integrated Linux kernel 3.5.0. We measured the application launch time in both cold start scenario and warm start scenario, where the main memory disk cache has stored none and all of the launch data, respectively. We also measured application launch time in a cold start with Mobifetch. Figure 1 shows that the average launch time reduction of Mobifetch is 28.1% over the cold start scenario without infill merge. In the figure, we can also see comparable improvements in the event of distance-based infill merge.

4. ONGOING WORK

We are currently porting Mobifetch to Linux-based Android and Ubuntu-ARM platforms to evaluate our scheme on widely-used smartphone platforms. We also plan to study fast prefetching methods by guaranteeing the number (or the size) of queued disk requests when the flash disk supports command queuing features, which the next standard of eMMC is expected to support.

5. REFERENCES

[1] Y. Joo, J. Ryu, S. Park, and K. G. Shin, FAST: Quick application launch on solid-state drives, In *Proceedings of the 9th USENIX FAST*, San Jose, California, USA, 2011.

[2] S. VanDeBogart, C. Frost, and E. Kohler, Reducing seek overhead with application-directed prefetching, In *Proceedings of the USENIX ATC*, San Diego, California, USA, 2009.