Usage Pattern-Based Prefetching: Quick Application Launch on Mobile Devices

Hokwon Song[1,2], Changwoo Min[1,2], Jeehong Kim[2], and Young Ik Eom[2]

[1] Samsung Electronics Co., Ltd., Suwon, Korea
[2] School of Information and Communication Engineering
Sungkyunkwan University, Suwon, Korea
{hokwon,multics69,jjilong,yieom}@ece.skku.ac.kr

Abstract. The startup time of applications is very important as a user perspective performance. If page faults occur frequently in the startup time, the user experience is subjected to an adverse effect. To reduce page faults, the prefetching scheme is used in the traditional OS. Previous studies proposed various schemes, but the most research was conducted for desktop PCs or special embedded devices. We propose the usage pattern-based prefetching scheme which is suitable to mobile devices. Therefore, this paper focuses on the user's applications usage patterns and the improvement of the startup time of application on mobile devices. To inspect the usage patterns, we collect the dataset of the application usage and then analyze collected data. Additionally, considering mobile devices which have relatively poor hardware resources, the lightweight prediction model is employed in the new scheme. The proposed scheme is implemented on both Android 2.2 and Linux kernel 2.6.29. It is tested on the emulator and evaluated by using the dataset. The startup time is improved about 5%, and the accuracy of the prediction is shown up to 59% for the practical dataset.

Keywords: Prefetching, Usage pattern, Mobile device.

1 Introduction

Recently, the embedded system has extended the coverage to the land of desktop PCs on the rapid development of hardware and software. Gradually, the change made users interested in the performance of mobile devices. The performance can be divided into the system performance, like CPU speed, memory size and display resolution, and the user perspective performance such as the startup time, the impression of a color and the user experience.

We focus on the improvement of the application's startup time. Although the startup time is one of the important user perspective performance factors, and the performance of hardware has evolved considerably compared to the previous, the startup time is still unsatisfying [11]. The main cause of the startup problem is the file IO [9, 11]. Since the file IO is much slower than CPU and main memory [7], and a task should be waiting until a page fault handler completes loading memory [8].

B. Murgante et al. (Eds.): ICCSA 2012, Part III, LNCS 7335, pp. 227–237, 2012.

Obviously, the startup time increases when the access to secondary storage to load pages occurs frequently. Therefore, the startup time will be dramatically improved if a page fault does not occur. In the OS field, the prefetching scheme is a traditional solution for reducing page faults [1, 2, 3, 10]. The number of page faults is decreased by loading pages in advance of launching the application.

This paper proposes the Usage Pattern-based Prefetching scheme for mobile devices called UPP. UPP predicts the next application based on the user's application usage patterns and fetches the memory pages of a predicted application in advance. Our study introduces the lightweight prediction model and the special triggering time for mobile devices. Mobile devices have more scanty resources than desktop PCs and a different lifetime of applications.

UPP is implemented on both Android 2.2 and Linux kernel 2.6.29 and evaluated by testing on the emulator.

This paper makes the following contributions:

- Observation of the application usage patterns between each application by using the practical dataset
- Suggestion of the lightweight prediction scheme
- Development of the Usage Pattern-based Prefetching called UPP
- Implementation and evaluation of UPP.

The paper is organized as fallows. In Section 2, we review other related work and discuss their efforts from the point of mobile devices. Section 3 analyzes the collected workload to understand the application usage patterns of users. In Section 4, we describe how to implement UPP, and provide the detail experimental setup and evaluate the experimental results. Finally, we conclude the paper and comment our future work in Section 5.

2 Related Work

2.1 Prefetching Scheme

In Linux and Windows, the prediction-based prefetching scheme is adopted for desktop PCs. The representatives are Preload [9] and SuperFetch [3]. They load the file-backed pages of an application which is expected to be executed in the near future. Thus, the prefetcher should monitor and analyze the user's access patterns. In case of [9], the prefetcher runs periodically to gather data and prefetch fault pages. Markov's probability model is employed to predict the next application. Additionally, they consider a multi-user environment.

The effectiveness of the prefetching scheme is influenced by the accuracy of prediction model. Scheduler-Assisted Prefetching [1] finds the next task by using a scheduler's queue. The pages of the next task are loaded into memory when the time quantum of current task is depleted until the base line. However, this scheme is proposed under the limited environment which frequently leads to swap-out due to heavy memory workload and does not consider the cold startup.

In studies of [4] and [5], they proposed a RT-PLRU scheme to find the optimal paging strategy. The scheme is focused on the time constraint of real time systems based on NAND flash. The NAND flash memory has been widely used as a secondary storage in the embedded systems. The RT-PLRU is a page replacement policy of the combination of pinning and LRU. The pinning scheme preloads pages and keeps them into main memory. It shows that the prefetching scheme contributes to satisfaction of real-time requirement even if the NAND-based system has the high read speed.

2.2 Smartphone Usages

Diversity in Smartphone Usage [6] is a comprehensive study of the smartphone use. The study found several characterized application usages of users activities. The application usage session described that users did not use installed applications as the same frequency. Specially, it is the same interest of our study. The authors analyzed the individual propensity of the user activities and their impact on energy consumption and use of network. We also collect the dataset of application usage and try to find patterns between the applications. Finally, the accuracy of the proposed prediction model is evaluated based on the dataset.

3 The Dataset Analysis

In this section, we analyze the traced data which is in order of launched applications on the smartphone. The relation between the applications will be inspected as application usage patterns.

3.1 The Dataset Gathering

Our work is based on the practical workloads which are collected by the monitoring applications on android mobile phones for a week. The application generates a log file in which the information of the launched applications is accumulated such as a start-time, the application's name and the binary's path. The dataset is summarized in Table 1.

Table 1. The overview of the dataset

User	# of applications launching	# of installed applications
User 1	220	82
User 2	389	128
User 3	191	69
User 4	317	207

3.2 Application Popularity

The number of installed applications and the number of launched applications for each user are shown in Fig.1 (a). Although the total installed applications are from 80 to 200, users were launched only partial applications from 15 to 45. Fig.1 (b) illustrates the application popularity ratio which is concentrated in 18~31% applications. The usage frequency of application is different, and only some applications are used concentrically. This usage pattern is similar with the research of [6].

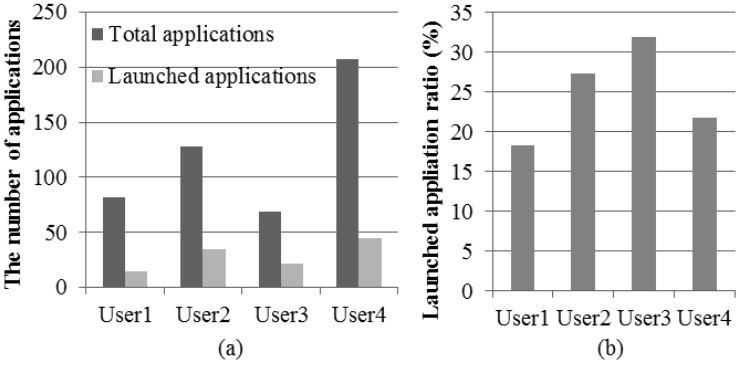

Fig. 1. (a) # of installed applications and # of launched applications for each user (b) The application popularity ratio

3.3 Application Usage Patterns

To find the user's application access patterns, the data is classified into the ordered list of applications which are consecutively launched after each application finishes. It is required to define the following notation.

- *T(Index application)*: the ordered list of traced applications which are consecutively executed after an Index application finishes.

For example, if a user sequentially executes applications like App1, App2, App3, App1, App3, App1, App1 and App4, the reorganized data are represented by *T(App1)={App2, App3, App1, App4}, T(App2)={App3}, T(App3)={App1,App1}* and *T(App4)=Null*. Fig.2 depicts how to collect each *T(App)* regarding the example.

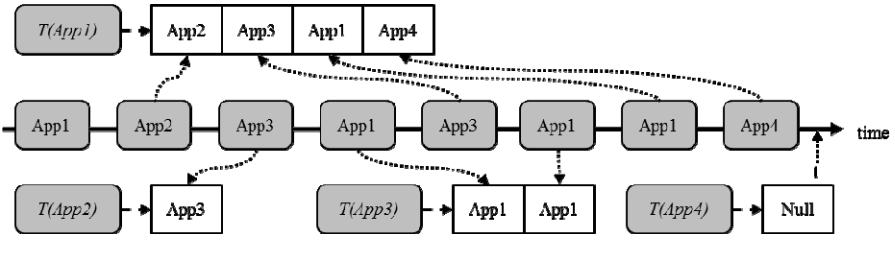

Fig. 2. Illustration of *T(App)* at time t

In case that the number of elements is smaller than four, it is difficult to find a meaningful pattern. Hence, only the *T(APP)* which is greater than four is analyzed to study the usage patterns. Fig. 3 and Fig. 4 illustrate the proportion of applications that starts after the application whose title is that of each graph, finishes. In Fig. 3, the user 1 executes Browser consecutively after using the Browser at the rate of 60% and launches MMS after Contacts was closed at the 55% chance. As shown in Fig. 4, the probability of beginning BeyondPod after the BeyondPod successively is 51%. KakaoTalk is expected to be used again after the KakaoTalk at the 53% rate. In the same way, the usage patterns of user 3 are launching Browser after Twitter and Facebook at the rate of 83% and 50% for respectively. The many higher relations between the applications are observed in other sets.

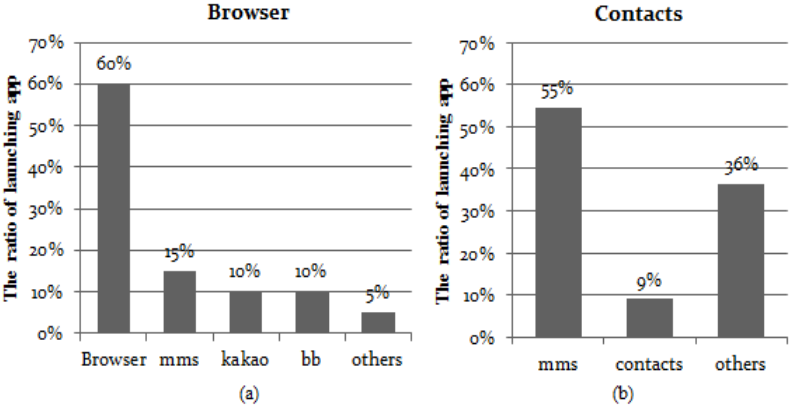

Fig. 3. Usage pattern of user 1

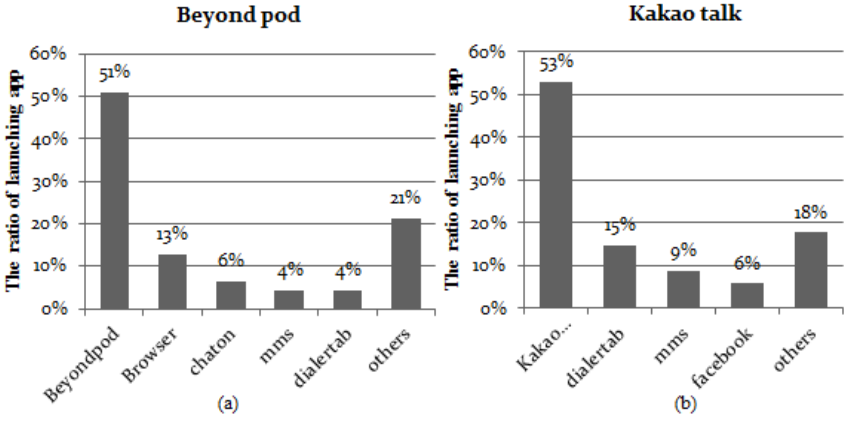

Fig. 4. Usage pattern of user 2

4 Design of UPP

4.1 The Application Prediction Model

The first-order Markov prediction model is applied to Behdad's Preload [9]. Although the Markov model is a renowned probability model, the more lightweight model is required for mobile devices since mobile devices have poor hardware resources compared to desktop PCs, and they are battery powered.

Considering the usage patterns observed in previous sessions, our prediction model requires some features like the following. Firstly, the information of the application that is consecutively executed is important. Secondly, the recency of the application use as well as frequency information should be reflected, since the usage pattern changes variously according to time. Finally, it should have low run-time overhead to reduce the power consumption and the burden of computation time.

We propose a Window and Weighted Sum-based prediction scheme called WWS that acquires a candidate based on the weighted sum of elements in the sliding window. The similar approach, a Window-based Direct Address Counting (WDAC) is adopted in the study [10].

The following notations are defined to explain the WWS mechanism.

- *W(Element application)*: the sum of element application by WWS.
- *C(Index application)*: the name of an element application which has the maximum *W(Element application)* among *T(Index application)*.

Fig. 5 depicts the algorithm on how to select the candidate when App1 finishes. App1 is the index application and the element applications are App3, App2, App3, App4, App3 and App5 sequentially. The window size is assumed to six. The results of *W(element)* are as follows; *W(App2)=0.4, W(App3)=1.8, W(App4)=0.8* and *W(App5)=1.2. C(App1)* is App3 which has the largest value. Therefore UPP loads the pages of App3, when App1 finishes.

When updating the list, the new comer is added at the end of window, and the head of window is deleted. To manage WWS, the memory is required as much as *window size * sizeof(Application identifier)* for each application. It is a small amount. The computation time is also negligible. Thus, WWS is a suitable scheme for mobile devices.

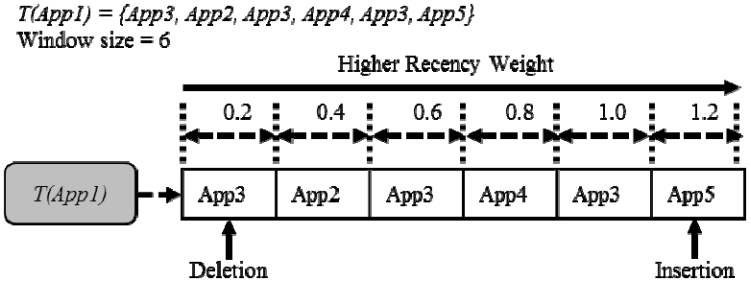

Fig. 5. WWS example

4.2 The Selection of Pages for Prefetching

We focus on accessing pages to induce file IO operations. This kind of access can be divided into two that are a major page fault that occurs when demand paging and explicit file IO call such as read and write. Thus, UPP gathers the information of pages from a major page fault and file IO calls during the application's startup time.

4.3 The Triggering Time of Prefetching

The application is launched through the main screen application (or called the home launcher) which has the entry points of all applications. In other words, the activating time of the main screen is the interval between applications. As shown in Fig. 6, the interval is enough to load fault pages. Thus, the prefetching is triggered when the main screen is activated.

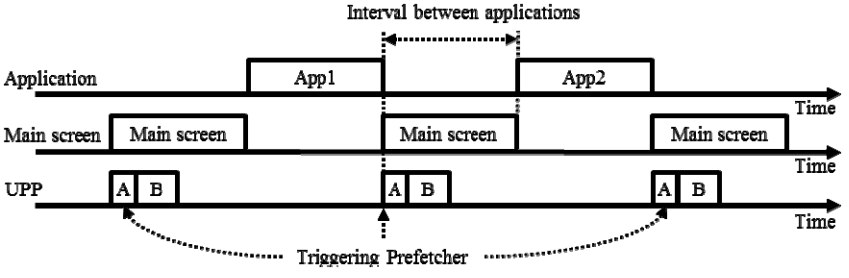

Fig. 6. The triggering time of Prefetching (A: Predicting the next application, B: Loading pages)

5 Evaluation

We explain the overall mechanism of UPP and discuss the results of experiment in this section. The goal of the experiment is to evaluate the effectiveness of WWS on the dataset and to improve startup time by the prefetching.

5.1 Implementation

UPP is composed of three components which are concreting prefetching lists, tracing application usage and prefetching. Fig. 7 illustrates the mechanism of UPP. The activity of concreting prefetching lists is represented by Arabic numbers (1~6). The capital letters (A, B) show the flow of tracing application usage which is *T(App)*. The prefetching activity is predicting a candidate application *C(App)* and loading memory pages that are referenced in the prefetching list. The small letters (a~c) are explained as the prefetching sequence. The activities are implemented on both Android 2.2 and Linux kernel 2.6.29.

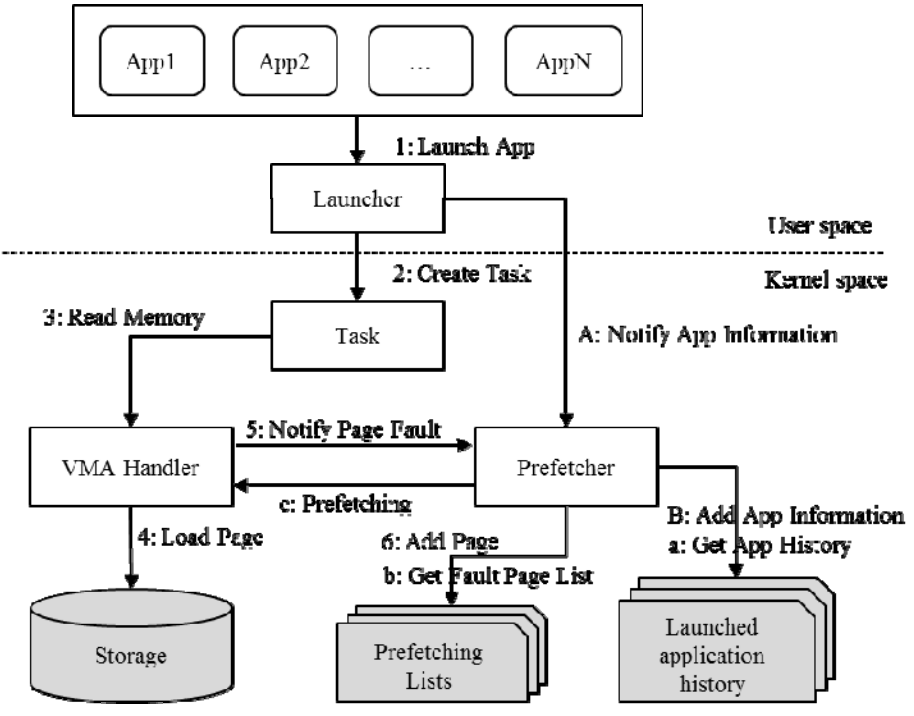

Fig. 7. UPP mechanism

5.2 Experimental Setups

Experiment for WWS

The accuracy of WWS is estimated by using the dataset of Table 1 in off-line. The window sizes are set to four and eight to observe the changes depending on various sizes.

Experiment for Prefetching

The prefetching is executed on an Android 2.2 emulator with Linux kernel 2.6.29. The size of a target application is 2.31 Mbytes. The android platform informs the time from the starting activity to the first exposed window at *windowsVisible()*. Hence, the time notified by the android is employed as the startup time. We use the average time of the multiple runs in cold startup for the prefetching and the no-prefetching.

5.3 Experimental Evaluation

WWS on the Dataset

We analyze the applications whose number of elements is greater than ten ($n(T(App)) > 10$). Fig. 8 (a) and Fig. 9 (a) are each user's application access pattern. Fig. 8 (b) and

Fig. 9 (b) depict the accuracy of WWS scheme according to the window size of four and eight. As shown in Fig. 8, the accuracy of WWS is at the rate of 54~59% if the access ratio concentrates on a few applications. The WWS is effective at the similar cases to Fig. 8 (a). Although a user launches various applications like Fig. 9 (a), we can succeed to predict the next application correctly about 20% as in Fig. 9 (b). A window size influences on the result of WWS slightly. Higher accuracy is shown in the narrow range when only partial elements of *T(App)* are repeatedly launched in the small set. Otherwise, if a few elements start repeatedly in the large set, better accuracy is resulted in. The optimal window size strongly relies on the usage patterns. For that reason, we leave finding on optimal window size as our future work.

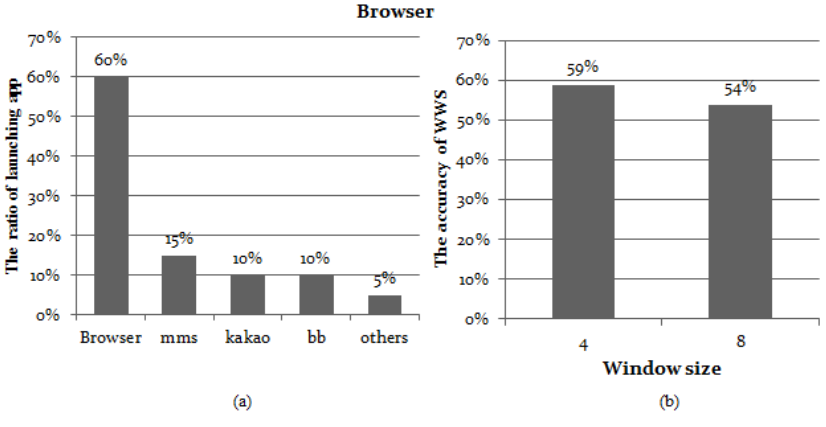

Fig. 8. (a) Usage pattern for Browser (b) WWS accuracy of Browser's candidate

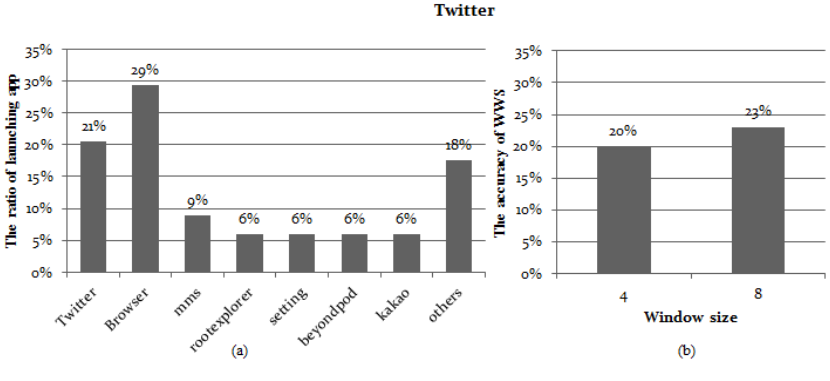

Fig. 9. (a) Usage pattern for Twitter (b) WWS accuracy of Twitter's candidate

Experiment for Prefetching

As shown in Fig. 10, the startup time with the prefetching is reduced by about 5% compared to the no-prefetching.

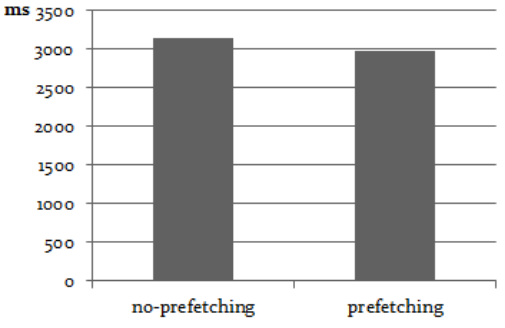

Fig. 10. Performance evaluation of the prefetching

6 Conclusion

Previously, we comment that the application startup time is important as the user perspective performance. If page faults occur frequently in the startup time, the user experience is subjected to an adverse effect.

Our study shows that users have their own usage patterns for each application. We propose a WWS scheme as a prediction model for mobile devices. WWS which reflects the frequency and the recency shows meaningful accuracy for the datasets. The scheme is low run-time overhead and low memory consumption. Thus, it is acceptable to mobile devices.

The improvement of the startup time by the prefetching component of UPP is smaller than our expectation since page faults still remain.

As our future work, we will study to find the best prediction model and optimize UPP.

Acknowledgments. This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2012-(H0301-12-3001))supervised by the NIPA(National IT Industry Promotion Agency).

References

1. Belogolov, S.A., Park, J., Hong, S.: Scheduler-Assisted Prefetching: Efficient Demand Paging for Embedded Systems. In: Proc. the 14th IEEE International Conference on Embedded and Real-Time Computation Systems and Applications (RTCSA), pp. 111–119 (2008)

2. Chiou, D., et al.: Scheduler-Based Prefetching for Multilevel Memories. MIT Computation Structures Group Memo 444 (2001)
3. Microsoft. Windows PC Accelerators, http://msdn.microsoft.com/en-us/windows/hardware/gg463388.aspx (updated: October 8, 2010)
4. Kim, J., Lee, D., Lee, C., Kim, K.: RT-PLRU: A New Paging Scheme for Real-Time Execution of Program Codes on NAND Flash Memory for Portable Media Players. IEEE Transactions on Computers 60(8) (2011)
5. Kim, J., Lee, D., Kim, K., Ha, E.: Real-Time Program Execution on NAND Flash Memory for Portable Media Players. In: Real-Time Systems Symposium, pp. 244–255 (2008)
6. Falaki, H., Mahajan, R., Kandula, S.: Diversity in Smartphone Usage. In: MobiSys (2010)
7. Compressed Caching for Linux: http://code.google.com/p/compcache/
8. Bovet, D.P., Cesati, M.: Understanding the Linux Kernel, 3rd edn. O'Reilly (2005)
9. Esfahbod, B.: Preload-An adaptive prefetching daemon. Master's thesis, Graduate Department of Computer Science, University of Toronto, Canada (2006)
10. Park, D., Du, D.H.C.: Mass Storage Systems and Technologies (MSST). In: 2011 IEEE 27th Symposium, pp. 1–11 (2011)
11. Joo, Y., Ryu, J., Park, S., Shin, K.G.: FAST: Quick Application Launch on Solid-State Drives. In: Proc. FAST 2011 Proceedings of the 9th USENIX Conference on File and Storage Technologies (2011)