ClusterFetch: A Lightweight Prefetcher for General Workloads

Haksu Jeong
Software R&D Center
Samsung Electronics
Suwon, 443-742, Korea
hks.jeong@samsung.com

Junhee Ryu
Networks Business
Samsung Electronics
Suwon, 443-742, Korea
junhee.ryu@samsung.com

Dongeun Lee
School of Electrical and
Computer Engineering, UNIST
Ulsan 689-798, Korea
eundong@unist.ac.kr

Jaemyoun Lee
Dept. Computer Science &
Engineering
Hanyang University
Ansan 426-791, Korea
jaemyoun@hanyang.ac.kr

Heonshik Shin
Dept. Computer Science &
Engineering
Seoul National University
Seoul 151-744, Korea
shinhs@snu.ac.kr

Kyungtae Kang
Dept. Computer Science &
Engineering
Hanyang University
Ansan 426-791, Korea
ktkang@hanyang.ac.kr

ABSTRACT

Application loading times can be reduced by prefetching disk blocks into the buffer cache. Existing prefetching schemes for general workloads suffer from significant overheads and low accuracy. *ClusterFetch* is a lightweight prefetcher that identifies continuous sequences of I/O requests and identifies the files that trigger them. The next time that the same files are opened, the corresponding disk blocks are prefetched. In experiments, ClusterFetch reduced the launch time, by which we refer to the latency that occurs when a program first runs, by 15.2 to 30.9%, and loading times, meaning the delays that are incurred while additional data is loaded from the disk during program execution, by 15.9%.

Categories and Subject Descriptors

D.4.3 [**Software**]: Operating Systems—*File Systems Management*

General Terms

Design

Keywords

ClusterFetch; Lightweight prefetch; Launch and loading times reduction

1. INTRODUCTION

Prefetching disk blocks effectively reduces subsequent disk access times, allowing applications to load and run more quickly [1, 2]. Successful prefetching depends on the accuracy with which upcoming disk I/O can be predicted, measured by the buffer cache hit-rate [3, 4], and many authors

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, TX, USA.
ACM 978-1-4503-3248-4/15/01.
http://dx.doi.org/10.1145/2668930.2688062

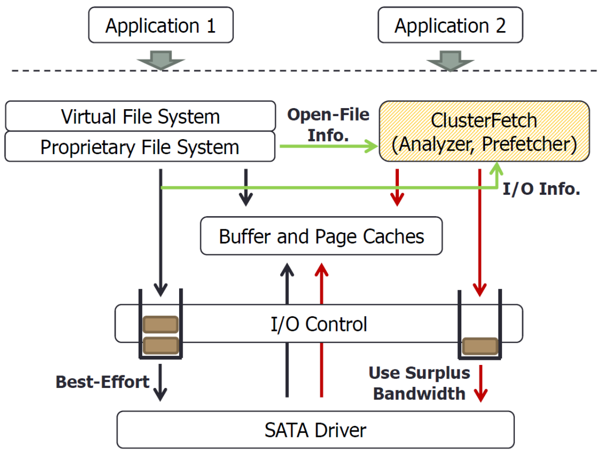

Figure 1: Operation of ClusterFetch.

have tried to improve this accuracy. However, most techniques incur significant memory and CPU overheads, and the predictions they make are not particularly accurate [3, 4]. In addition, existing prefetching approaches have largely focused on reducing application *launch time*, which is the delay in starting a program.

ClusterFetch is a *general-purpose*, *lightweight* prefetcher that runs within the Linux kernel to reduce the launch time, and also the delays incurred when a program which is already running has to load additional data from disk (i.e. *loading times*). Experiments show reductions of up to 30.9% in launch times and 15.9% in loading times, at the cost of less than 3MB of memory overhead in establishing correlations between disk blocks.

2. DESIGN AND IMPLEMENTATION

Periods during which there is continuous disk I/O, but negligible memory and CPU activity, can be identified by using a circular queue to record every disk I/O operation. For as long as the period between the first and the last entry in the queue is less than a predefined threshold, ClusterFetch considers the entries to correspond to a period of continuous disk I/O, and stores the identification numbers (IDs) of the blocks that were accessed in a prefetch information file. In addition, ClusterFetch traces the file which triggered the continuous I/O by looking at the log of file opening operations. Then it links this trigger file to an entry in the prefetch information file by setting the sticky permission bit on the trigger file; this bit is available because the current implementation of Linux only uses it when accessing a directory file. Subsequently, whenever the Linux kernel opens the trigger file, the disk blocks corresponding to the IDs written in the prefetch information file are brought into the buffer cache. Thus this scheme is able to detect, and utilize, the correlation between disk blocks with a negligible overhead, unlike many previous prefetching schemes which impose significant memory overhead [4] or generate limited information on block correlation [3].

To avoid prefetching operations delaying I/O from other processes, I/O control process shown in Figure 1 manages the I/O priority of other block requests and prefetch operations. In addition, ClusterFetch provides a control parameter to limit the I/O bandwidth of prefetching operations. ClusterFetch also uses native command queuing (NCQ) within the SATA2 standard to maximize I/O throughput. The structure and operation of ClusterFetch are illustrated in Figure 1.

3. EXPERIMENTAL RESULTS AND CONCLUSION

We compared launch and loading times of three popular applications, with and without ClusterFetch. The applications were Eclipse (a development tool), Flightgear (a flight simulator), and Savage 2 (a game), and we measured both cold and warm start times. The results in Table 1 show that our scheme reduce the launch times of Eclipse and Flightgear by 30.9% and 15.2% respectively, and the loading time of Savage 2 by 15.9%, at the expense of the minor overhead incurred in detecting and utilizing the correlation between disk blocks.

Table 1: Effect of ClusterFetch on Launch and Loading Times

Applications	Cold	Warm	ClusterFetch	Reduction
Eclipse (LA)	16.5s	6.1s	11.4s	30.9%
Flightgear (LA)	27.5s	18.9s	23.3s	15.2%
Savage 2 (LO)	22.0s	17.0s	18.5s	15.9%

LA: Launch, LO: Loading

4. ACKNOWLEDGMENTS

This research was supported in part by the Ministry of Science, ICT, and Future Planning (MSIP), Korea, under the Information Technology Research Center support program (NIPA-2014-H0301-14-1044) supervised by the NIPA (National ICT Industry Promotion Agency), in part by the MSIP, Korea, under the IT/SW Creative research Program supervised by the NIPA (Nationa1 IT Industry Promotion Agency) (NIPA-2013-H0502-13-1061), and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the MSIP (NRF-2013R1A1A105-9188).

5. REFERENCES

[1] JOO, Y., RYU, J., PARK, S., AND SHIN, K. G. Fast: Quick application launch on solid-state drives. In *Proc. FAST'11* (2011), pp. 259–272.

[2] YAN, T., Chu, D., Ganesan, D., Kansal, A., AND Liu, J. Fast app launching for mobile devices using predictive user context. In Proc. *MobiSys'12* (2012), pp. 113–126.

[3] DING, X., JIANG, S., CHEN, F., DAVIS, K., AND ZHANG, X. Diskseen: Exploiting disk layout and access history to enhance I/O prefetch. In *Proc. ATC'07* (2007), pp. 261–274.

[4] LI, Z., CHEN, Z., SRINIVASAN, S. M., AND ZHOU, Y. C-miner: Mining block correlations in storage systems. In *Proc. FAST'04* (2004), pp. 173–186.