P2Cache: An Application-Directed Page Cache for Data-Intensive Applications

Dusol Lee
Seoul National University

Inhyuk Choi
Seoul National University

Chanyoung Lee
Seoul National University

Sungjin Lee
DGIST

Jihong Kim
Seoul National University

Modern data-intensive applications [1, 2] require a large amount of data movements between an SSD and the DRAM memory of a host system. In order to efficiently manage such slow data transfers, most operating systems employ page caches in the host DRAM memory that exploits the reference locality of I/O accesses. Although OS-level page caches were widely used for improving the I/O performance of various applications, their performance improvements are often disappointing because the cache management policies of a page cache may not match well with the specific I/O characteristics of an application. For example, when an application repeatedly accesses a large range of I/O addresses in a looping pattern, the MRU (Most Recently Used) replacement policy may outperform the commonly used LRU (Least Recently Used) replacement policy. If an application accesses the I/O address space randomly, the standard read-ahead policy used by the page cache may prove ineffective.

In order to mitigate the mismatch problem between the I/O characteristics (of an application) and the policies (of a kernel-level page cache), data-intensive applications often implement their own page caches at the application level [2, 3]. However, supporting a user-level page cache at the application level presents several practical challenges.

Even if a user-level page cache is implemented to avoid utilizing a kernel-level page cache (e.g., as in RocksDB's Direct-IO [3]), when applications oversubscribe their cache memory, leading to memory pressure, the OS will evict memory containing application data through undesirable swap I/O activities. Such swapping activity by the kernel, which disregards the I/O semantics of the application, degrades application performance due to low cache utilization. Additionally, it may not be able to fully take advantage of certain kernel-supported functions, such as those for ensuring data protection and consistency. Moreover, if there are significant changes in either the SSD or the host memory system, a user-level cache would need to be re-implemented.

In this paper, we propose an *application-directed* kernel-level page cache, P2Cache (Programmable Page Cache), that allows an application developer to build a *custom kernel-level* page cache that matches the I/O characteristics of a target application. P2Cache aims to leverage the advantages of a

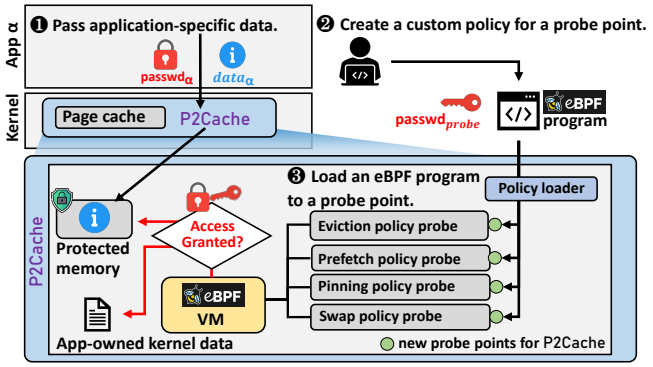
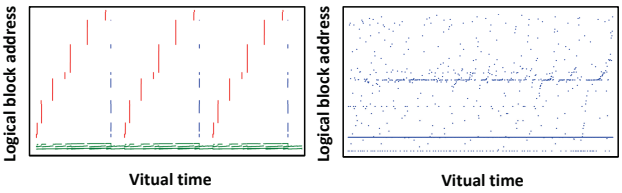


Figure 1: An operational overview of P2Cache.

kernel-level cache by enhancing the existing Linux page cache with four additional probe points. These probe points enable the integration of user-level customization through eBPF programs [4] with the Linux kernel-level page cache. Using a P2Cache-specific API, P2C API, application developers can create customized kernel-level page caches.

Figure 1 shows an operational overview of P2Cache. To create a custom page cache for an application α, application-specific data $\texttt{data}_\alpha$ of the app α may be required for developing a new cache policy. If they are needed, $\texttt{data}_\alpha$ is moved to the kernel's protected memory (❶). To avoid unauthorized accesses to $\texttt{data}_\alpha$ from other applications, a secret password ($\texttt{passwd}_\alpha$) is assigned for each custom page cache. Using P2C API functions along with the kernel data for the app α, an eBPF program is implemented for each probe point of P2Cache (❷). After loading eBPF programs into their respective probe points (❸), a custom page cache for the app α becomes effective by executing the eBPF programs whenever the kernel's execution flow reaches those points. Before an eBPF program is executed, an extended eBPF VM verifies the program's authorization to access kernel data owned by the application α as well as application-specific data $\texttt{data}_\alpha$. This is achieved by comparing the $\texttt{passwd}_{probe}$ of the eBPF program with the $\texttt{passwd}_\alpha$ passed from the corresponding application.

To evaluate the effectiveness of P2Cache, we implemented it on a Linux server equipped with a high-performance 2TB NVMe SSD, running kernel version 5.8. For benchmarking,



(a) Lumos. (b) GraphWalker.

Figure 2: I/O patterns of Lumos and GraphWalker.

we selected two open-source, data-intensive graph processing frameworks: Lumos [1], which leverages the OS page cache, and GraphWalker [2], which utilizes its own user-level cache. Recent graph processing engines enhance I/O efficiency with specialized structures and optimizations, but mismatched application I/O semantics and kernel page cache behavior can hinder performance. By benchmarking these applications, we aimed to identify and address areas for improvement. We used the Twitter graph dataset [5] for the evaluation, and the performance metrics are normalized to the optimal case where the memory size is large enough.

Figure 2 plots I/O reference pattern of the two graph applications, Lumos [1] which executes Pagerank [6] algorithm and GraphWalker [2] which executes Simrank [7] algorithm. As shown in Figure 2(a), Lumos maintains multiple files and scans them simultaneously, sending mixed I/O patterns to the disk. On the other hand, as shown in Figure 2(b), the I/O reference patterns of GraphWalker are mostly random. This is because GraphWalker dynamically selects and evicts subgraphs with low walk counts at the moment of eviction.

To enhance the performance of Lumos, we applied a customized MRU eviction policy along with a custom pinning policy which retains metadata files in the page cache. As shown in the Figure 3, the performance enhancement of Lumos, utilizing a custom page cache implemented with P2Cache, results in up to a 32% improvement compared to Baseline. Additionally, when compared to MRU+PIN, P2Cache consistently exhibits performance gains of up to 15%. Among two policy optimizations applied for Lumos, the customized MRU eviction policy, which differentiates the importance of accessed files for each iteration, has a distinction from the basic MRU policy.

To enhance GraphWalker's performance using P2Cache, we adopted an optimization strategy from a user-level page caching technique employed in GraphWalker, where blocks with the lowest walk counts are evicted first [2]. To replicate this approach, we transferred the walk-num values for each block to the kernel after every iteration step and implemented a custom eBPF program for the swap policy probe. As shown in the Figure 4, GraphWalker, enhanced with a custom page cache using P2Cache, achieves up to 14% better performance than Baseline, even with swap policy customization alone. Under the default Baseline page cache,

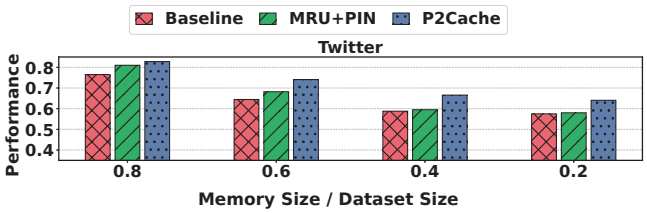


Figure 3: Lumos performance comparisons among the default kernel page cache, simple optimization, and custom page cache.

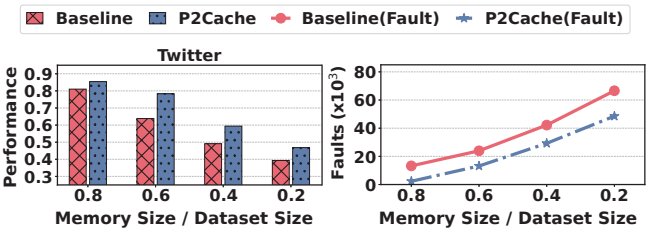


Figure 4: GraphWalker performance comparisons between the default kernel page cache and custom page cache.

GraphWalker suffers significant performance loss due to frequent page evictions and increased page faults as memory becomes limited.

We presented P2Cache, an application-directed kernel-level page cache for Linux systems that enables developers to create custom caches tailored to I/O workloads. P2Cache extends the existing kernel page cache by introducing four new probe points, retaining its benefits. Our evaluation shows that P2Cache effectively enhances I/O performance in data-intensive applications like graph processing.

Original Publication. Dusol Lee, Inhyuk Choi, Chanyoung Lee, Sungjin Lee, and Jihong Kim. P2cache: An application-directed page cache for improving performance of data-intensive applications. In Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage), 2023. DOI: https://doi.org/10.1145/3599691.3603408

REFERENCES

[1] Keval Vora. LUMOS: Dependency-Driven Disk-based Graph Processing. In *Proceeindgs of the USENIX Annual Technical Conference (ATC)*, 2019.

[2] Rui Wang, Yongkun Li, Yinlong Xu Hong Xie and, and John C. S. Lui. GraphWalker: An I/O-Efficient and Resource-Friendly Graph Analytic System for Fast and Scalable Random Walks. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2020.

[3] EighteenZi - GitHub. Direct-IO.md. https://github.com/EighteenZi/rocksdb_wiki/blob/master/Direct-IO.md, 2017.

[4] eBPF. eBPF Documentation. https://ebpf.io/what-is-ebpf/, 2022.

[5] Stanford University. Twitter follower network. https://snap.stanford.edu/data/twitter-2010.html, 2010.

[6] Wikipedia. PageRank. https://en.wikipedia.org/wiki/PageRank, 2023.

[7] Glen Jeh and Jennifer Widom. SimRank: A Measure of Structural-Context Similarity. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2002.