CrossPrefetch: Accelerating I/O Prefetching for Modern Storage

S. Garg et al.

ASPLOS '24

2025. 02. 05

Presented by SeongHyeon Lee
leesh0812@dankook.ac.kr

DANKOOK UNIVERSITY

Dankook University
System Software Laboratory

Outline

- **Outline**
  - **Introduction & Background**
  - **Motivation**
  - **Crossprefetch**
  - **Evaluation**
  - **Conclusion**

Introduction & Background

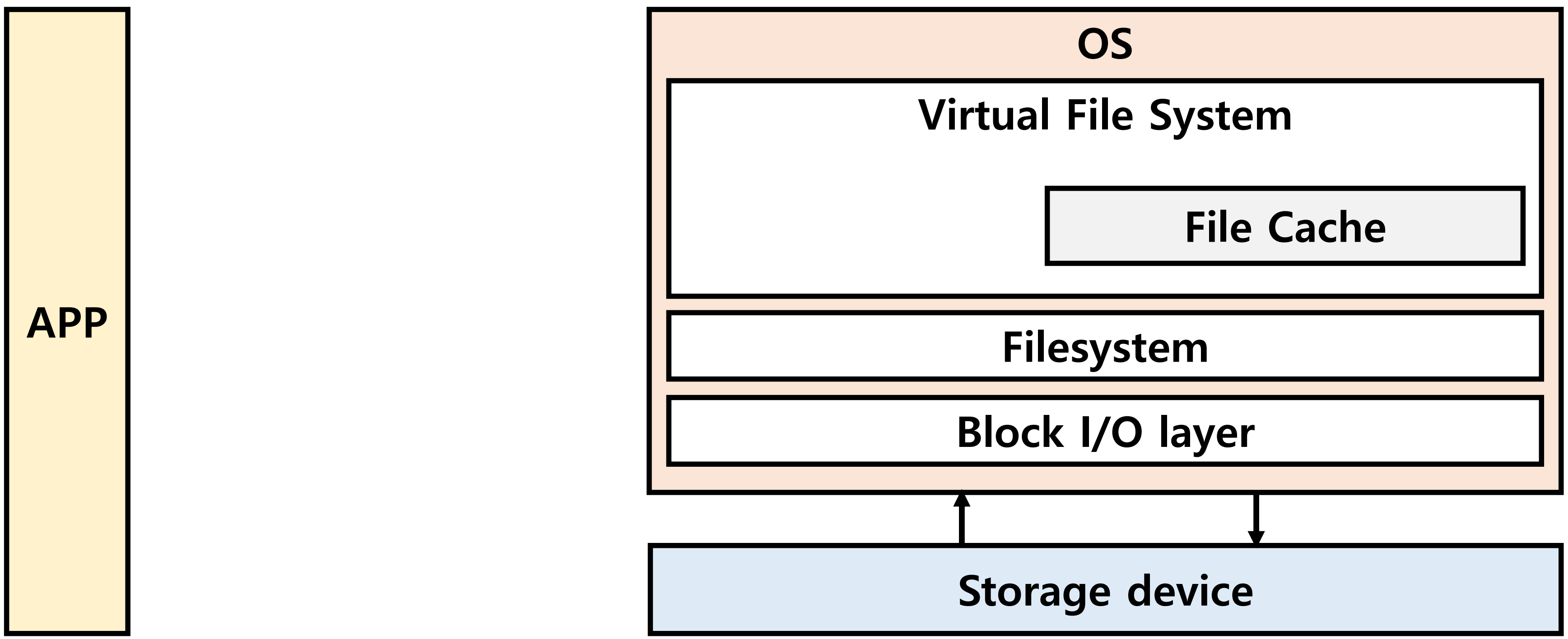
- **Storage hardware is getting faster**
  - Newest NVMe SSDs have 7-10 GB/s throughput for sequential reads.

- **Applications have complex I/O access patterns**
  - Large storage bound apps transition between sequential, stride and random access pattern

- **Many filesystems specifically for modern storage hardware**
  - F2FS, noova, XFS etc.

Introduction & Background

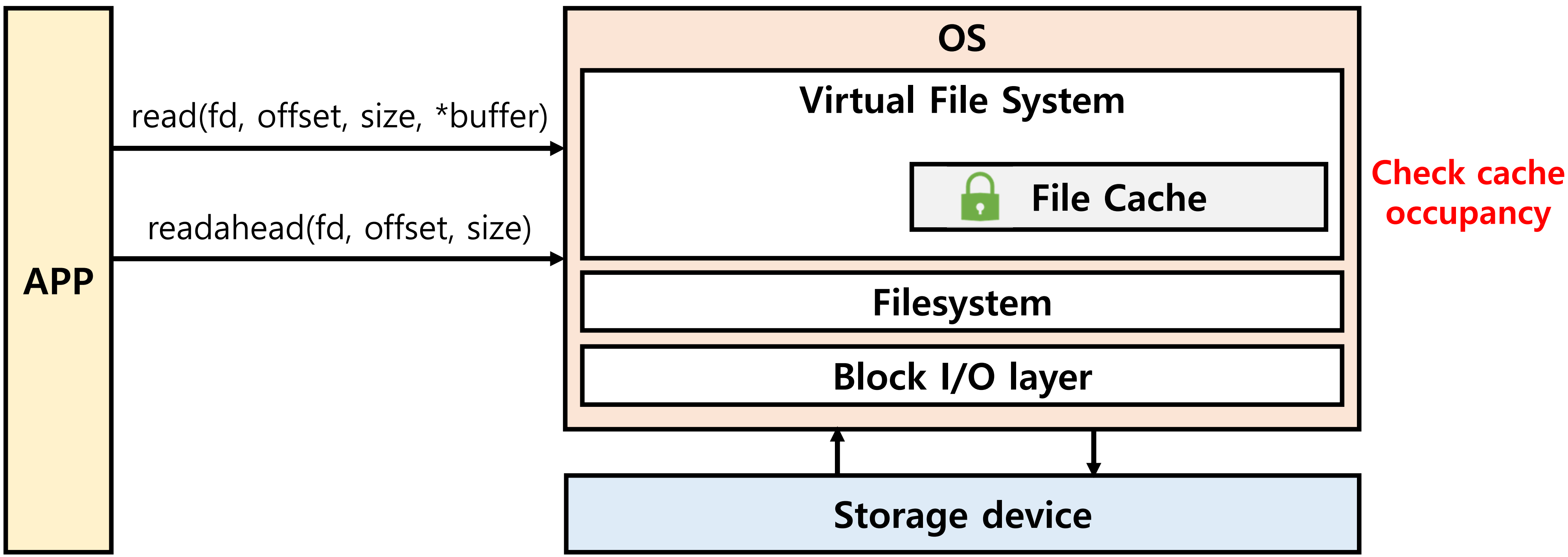
- **Filesystem workflow**



Introduction & Background

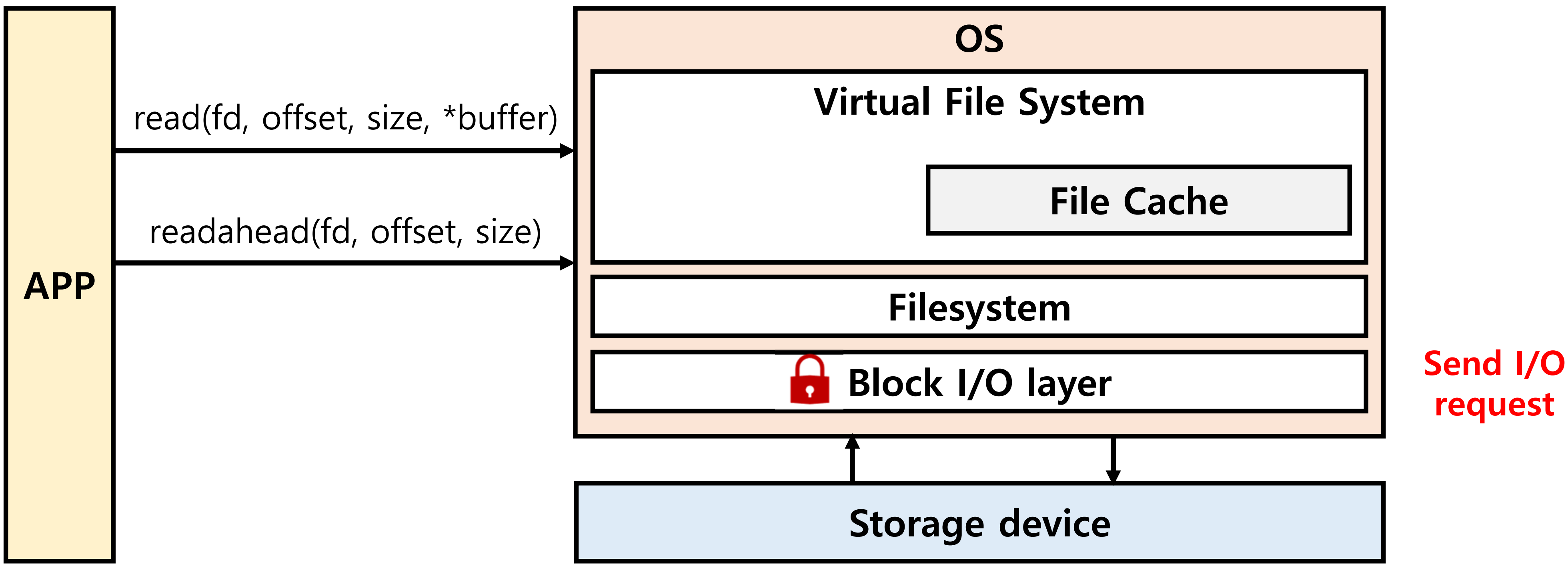
Readers Lock
Writers Lock

- **Filesystem workflow**



Introduction & Background

- **Filesystem workflow**

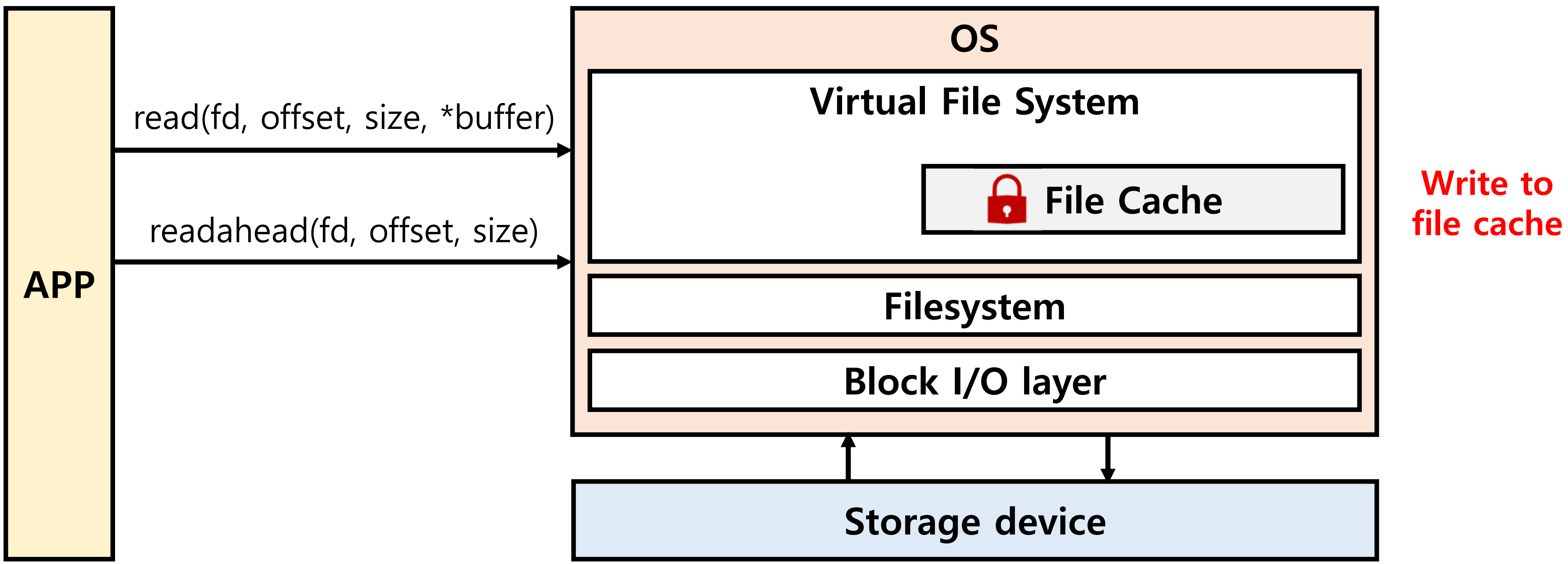


Introduction & Background

Readers Lock

Writers Lock

- **Filesystem workflow**

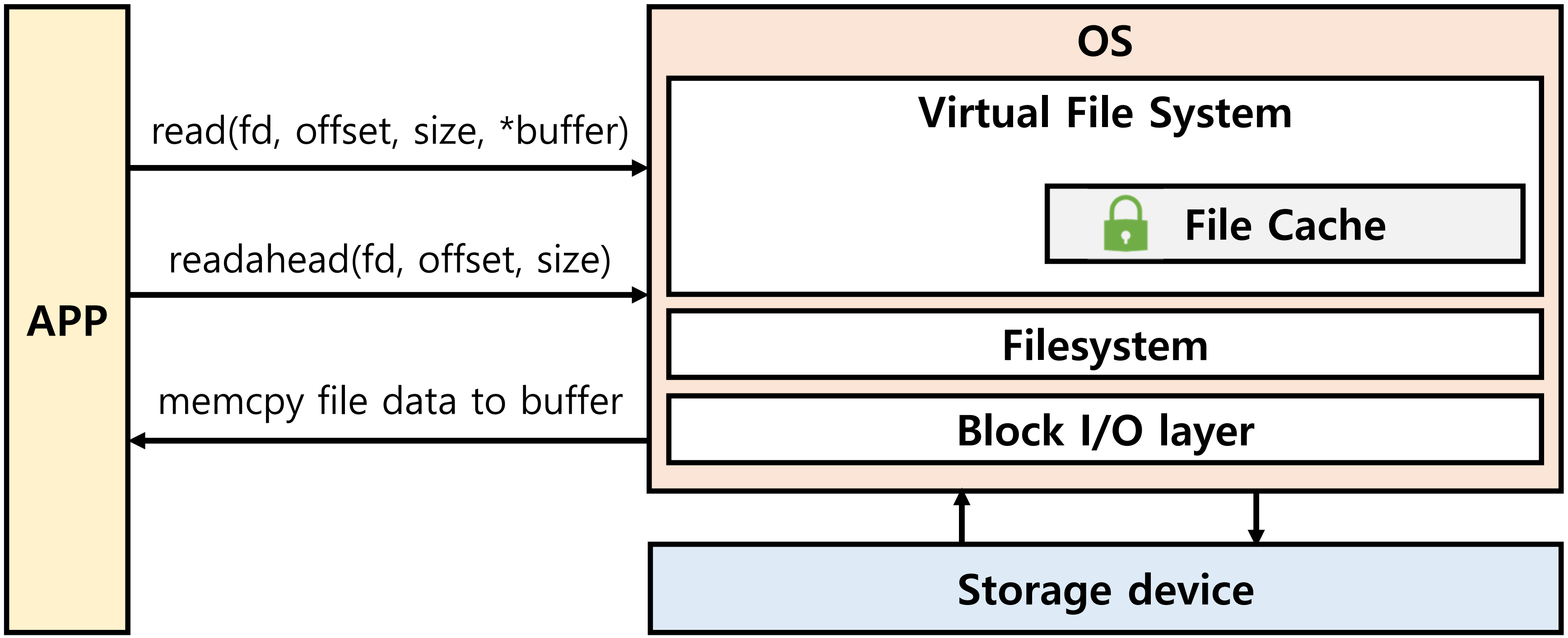


Introduction & Background

Readers Lock

Writers Lock

- **Filesystem workflow**

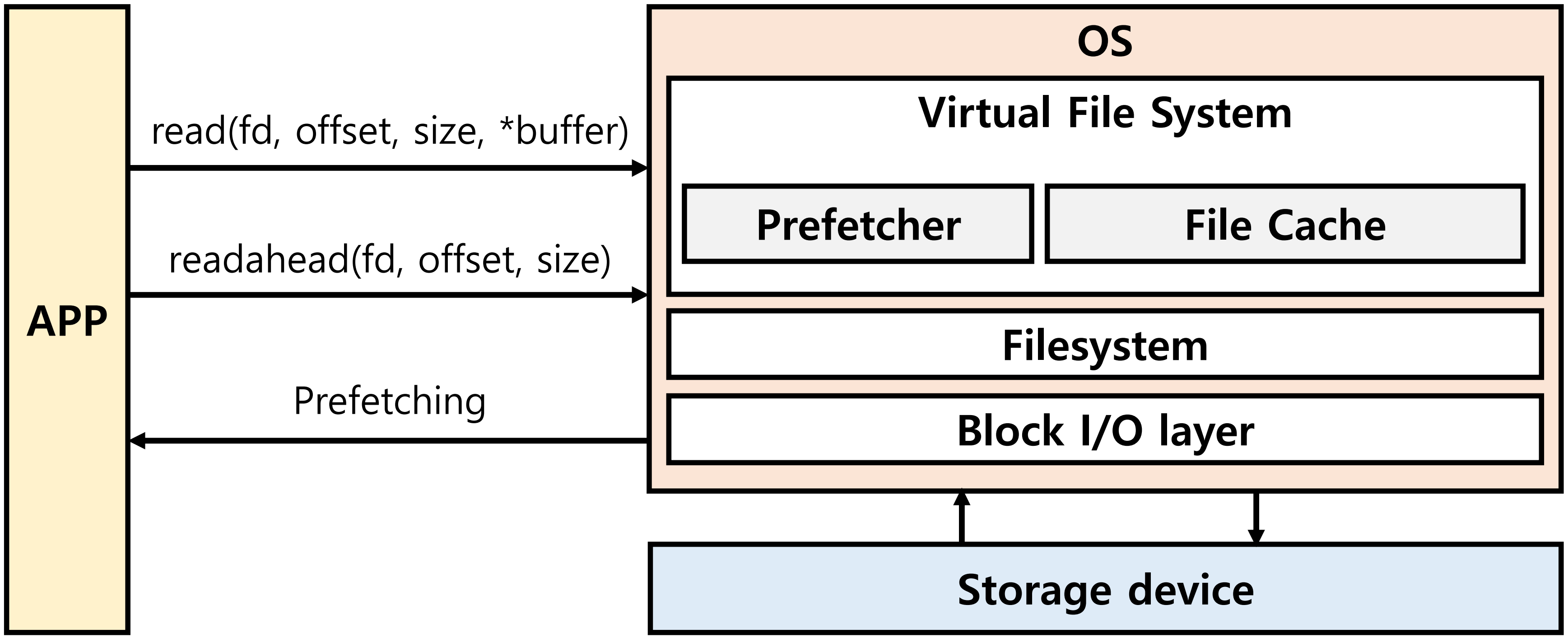


Introduction & Background

Readers Lock

Writers Lock

- **Filesystem workflow**



Introduction & Background

- **System call**
  - int **fadvise**(int fd, off_t offset, off_t len, int advice)
  - int **readahead**(int fd, off64_t offset, size_t count)
- **Advise**
  - I/O access pattern : sequential, random
  - Prefetching : will need, won't need

No information on what was prefetched to the application

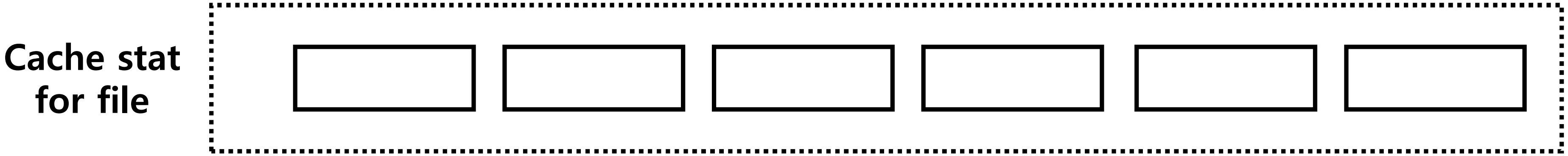
→Lack of cache visibility leads to poor prefetching and I/O performance

Introduction & Background

App requested for file page

File page in cache

- **Lack of I/O cache visibility : Under prefetching**



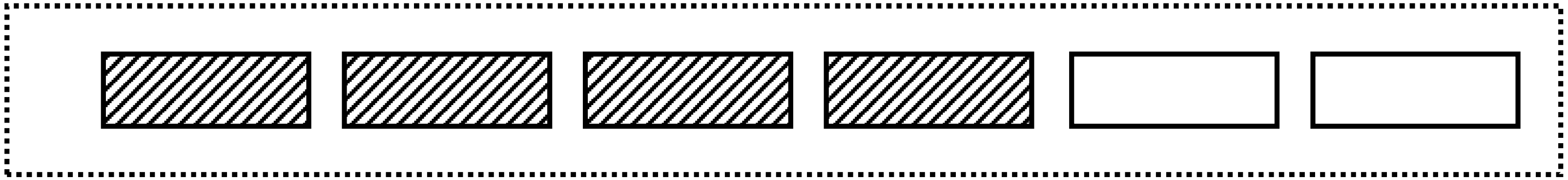
Introduction & Background

App requested for file page
File page in cache

- **Lack of I/O cache visibility : Under prefetching**

Request 1: readahead(f1, start: pg1, sz: 4pg)

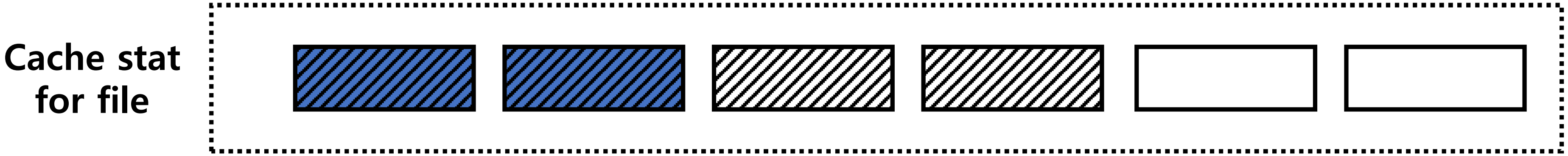
Cache stat for file



Introduction & Background

- **Lack of I/O cache visibility : Under prefetching**

Request 1: readahead(f1, start: pg1, sz: 4pg)

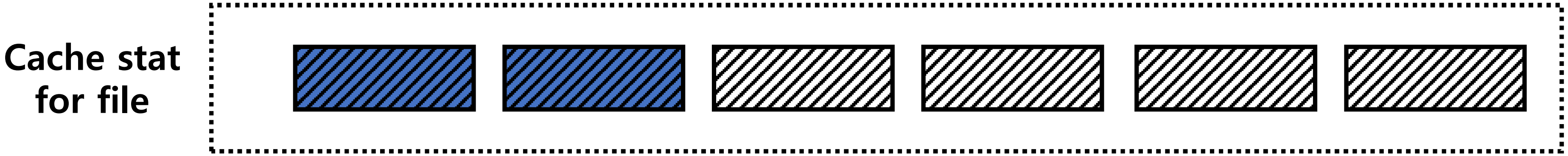


Request 1: Prefetched 2 pages

Introduction & Background

- **Lack of I/O cache visibility : Under prefetching**

Request 1: readahead(f1, start: pg1, sz: 4pg)   **Request 2:** readahead(f1, start: pg5, sz: 2pg)



Request 1: Prefetched 2 pages

Introduction & Background

App requested for file page

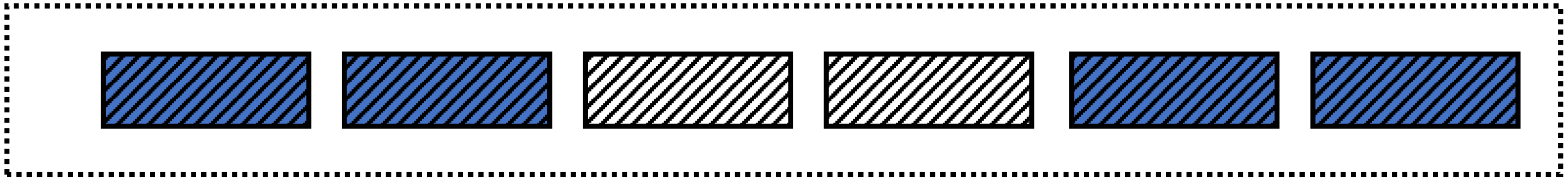
File page in cache

- **Lack of I/O cache visibility : Under prefetching**

Request 1: readahead(f1, start: pg1, sz: 4pg)

Request 2: readahead(f1, start: pg5, sz: 2pg)

Cache stat for file



Request 1: Prefetched 2 pages

Request 2: Prefetched 2 pages

Introduction & Background

App requested for file page

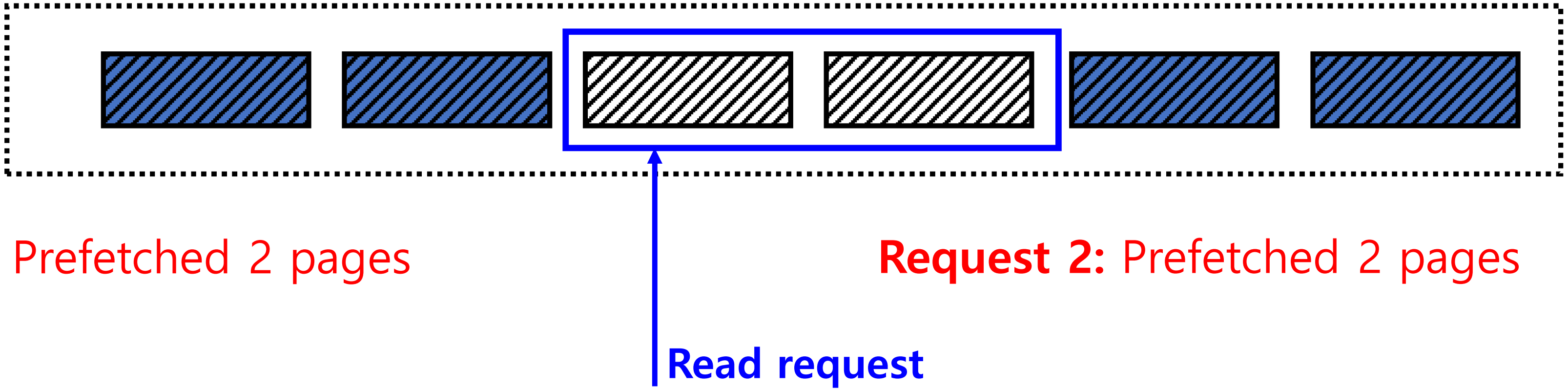
File page in cache

- **Lack of I/O cache visibility : Under prefetching**

Request 1: readahead(f1, start: pg1, sz: 4pg)

Request 2: readahead(f1, start: pg5, sz: 2pg)

Cache stat for file



Request 1: Prefetched 2 pages

Request 2: Prefetched 2 pages

Cache Miss

Introduction & Background

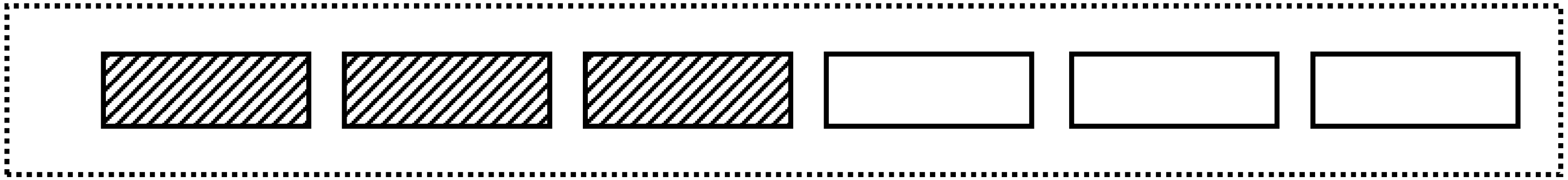
App requested for file page

File page in cache

- **Lack of I/O cache visibility : Over prefetching**

Request 1: readahead(f1, start: pg1, sz: 3pg)

Cache stat for file



Introduction & Background

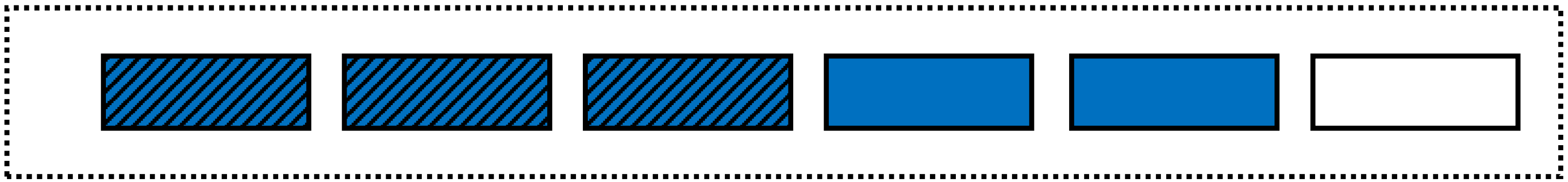
App requested for file page

File page in cache

- **Lack of I/O cache visibility : Over prefetching**

Request 1: readahead(f1, start: pg1, sz: 3pg)

Cache stat for file



Introduction & Background

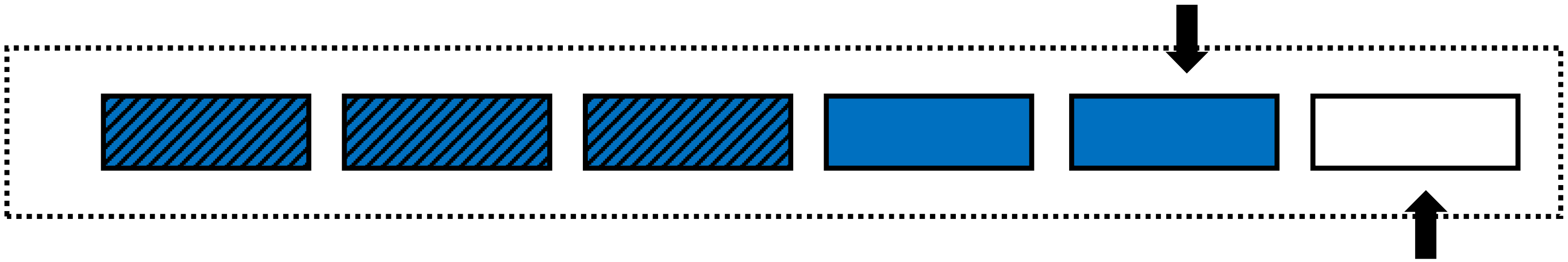
App requested for file page

File page in cache

- **Lack of I/O cache visibility : Over prefetching**

Request 1: readahead(f1, start: pg1, sz: 3pg) **Request 2:** readahead(f1, start: pg5, sz: 1pg)

Cache stat for file



Read(f1, start: pg6, sz: 2KB)

Introduction & Background

- **Related work**

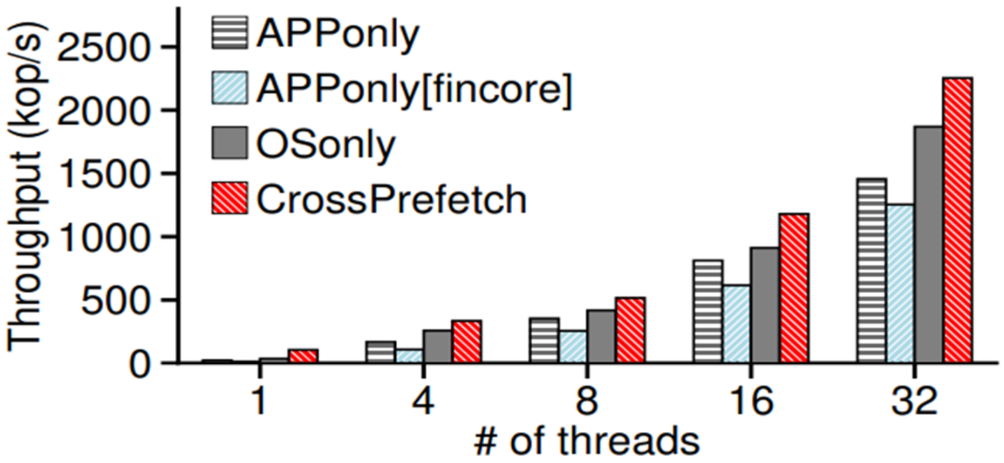
Prefetching Method	Description	Limitations
OS-based	Uses readahead(), fadvise(), and madvise() for prefetching	Applications cannot verify if prefetching was performed, rigid policies
Application-based	Prefetching implemented within applications like RocksDB, MySQL	Cannot check OS cache state, increases development complexity
Machine Learning (ML)-based	Uses Markov Chains, Reinforcement Learning (RL) for prefetching	Requires training time, lacks OS-application coordination
Compiler-based	Optimizes prefetching through static analysis	Struggles with dynamic workload changes, lacks OS cooperation

Motivation

- **Lack of Awareness of Prefetching Status**
  - Applications cannot verify if prefetching was successful, leading to redundant or inefficient requests.
- **Concurrency Bottlenecks & Software Overhead**
  - Multiple threads sharing the same file can cause lock contention (e.g., Xarray lock in Linux), increasing system call overhead.
- **Inefficient Memory Utilization**
  - Linux's incremental prefetching (128KB limit) does not adapt dynamically to available memory, leading to cache misses.

Motivation

- **RocksDB Analysis**



Osonly
where the OS handles prefetching
APPonly
disables OS and application prefetching for random access
APPonly[fincore]
uses a background prefetching thread to use fincore

	APPonly	APPonly[fincore]	OSonly	CrossPrefetch
Locking (%)	16	34	27	19
Cache Misses (%)	98.2	91.5	84.3	63.7

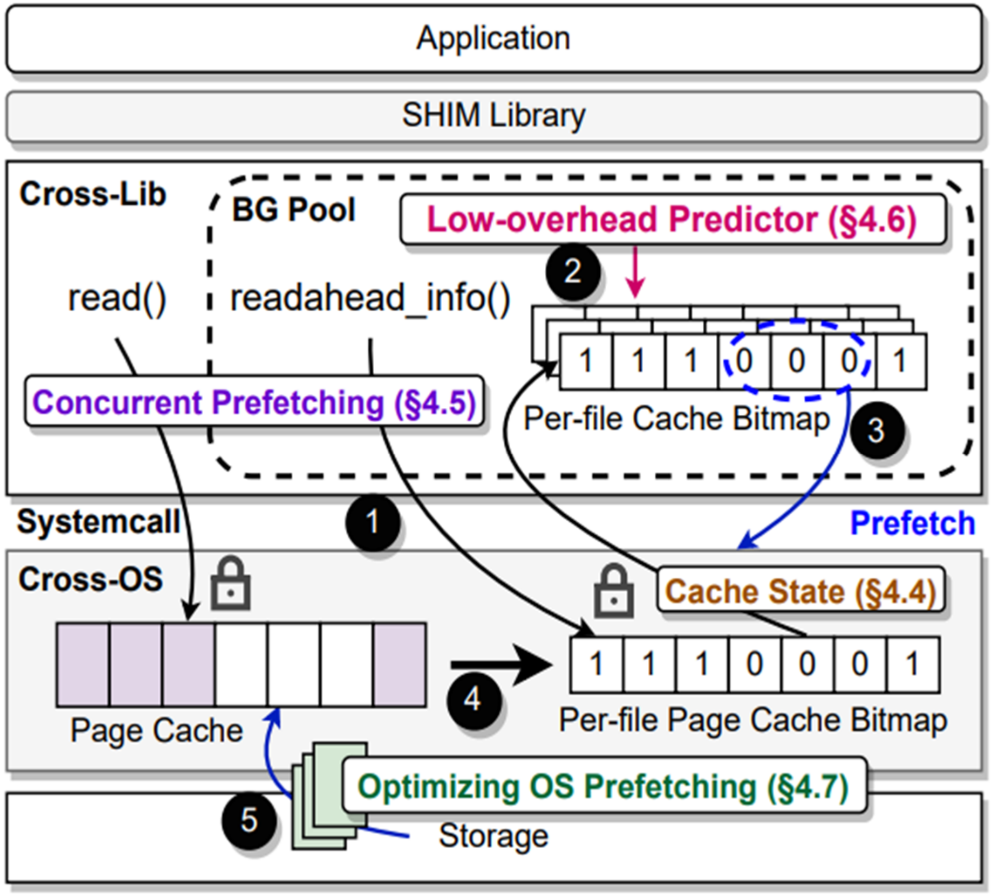
Table 1. Lock Overhead and Avg. Cache Misses (in %)

CrossPrefetch

- **Disaggregate I/O prefetching responsibilities between the OS and a user-level runtime.**
- **Support concurrent prefetching and lightweight prediction.**
- **Enable aggressive prefetching and eviction without impacting memory budget.**

CrossPrefetch

- **Cross-layered Approach**
  - **Cross-OS**
    - Unlike traditional OS methods, manages **Per-file Page Cache Bitmap** at the inode level and optimizes prefetching based on this information.
  - **Cross-Lib**
    - Allows applications to directly control prefetching.

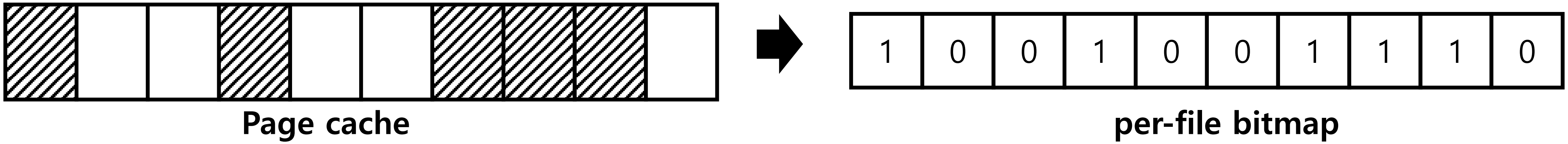


CrossPrefetch

- **Provide Visibility on I/O Prefetching State**
  - Overhead of scanning OS cache data structures
  - Cost of copying page cache state from OS to user space
  - Need for fast updates to per-file cache bitmap

CrossPrefetch

- **Provide Visibility on I/O Prefetching State**
  - per-file bitmap



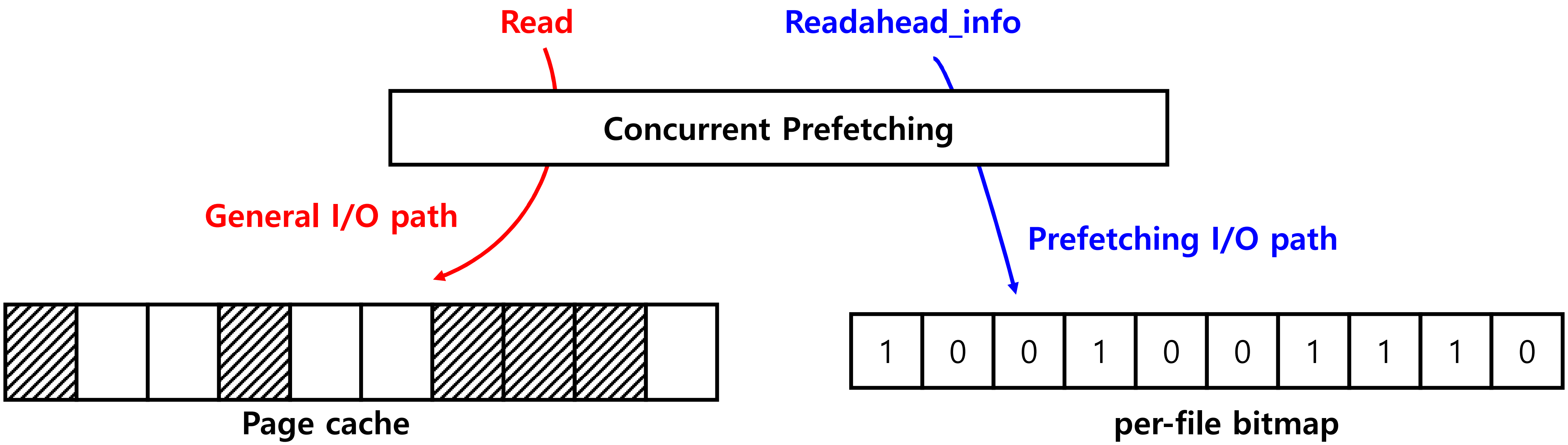
  - readahead_info systemcall

int **readahead**(int fd, off64_t offset, size_t count)

Int readahead_info(int fd, off64_t offset, size_t count, struct ra_info * info)

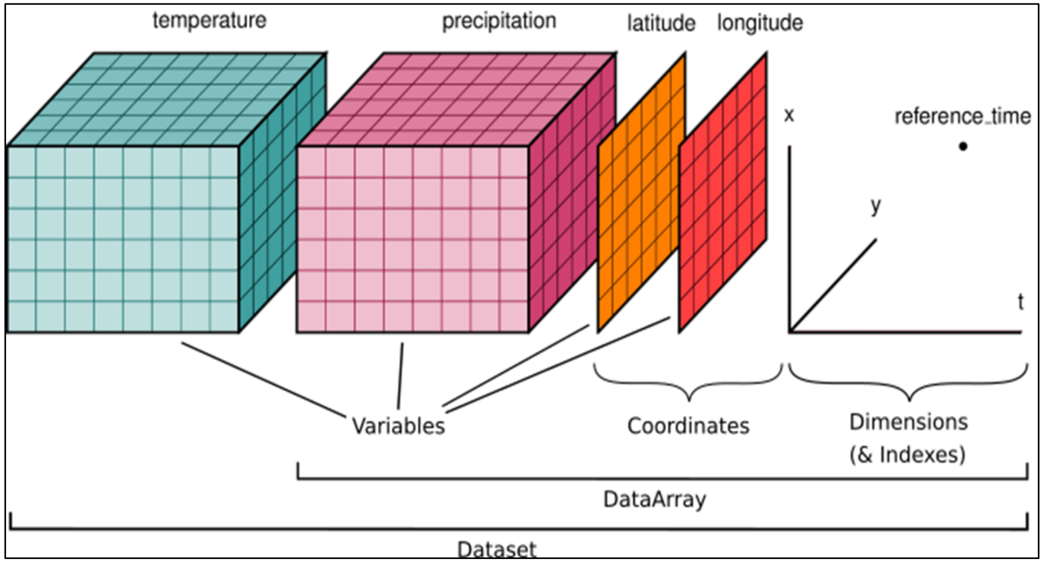
CrossPrefetch

- **Provide Visibility on I/O Prefetching State**
  - Utilize per-file bitmaps to track cache status and improve prefetching efficiency
  - Guarantee concurrency by separating the fast path and slow path

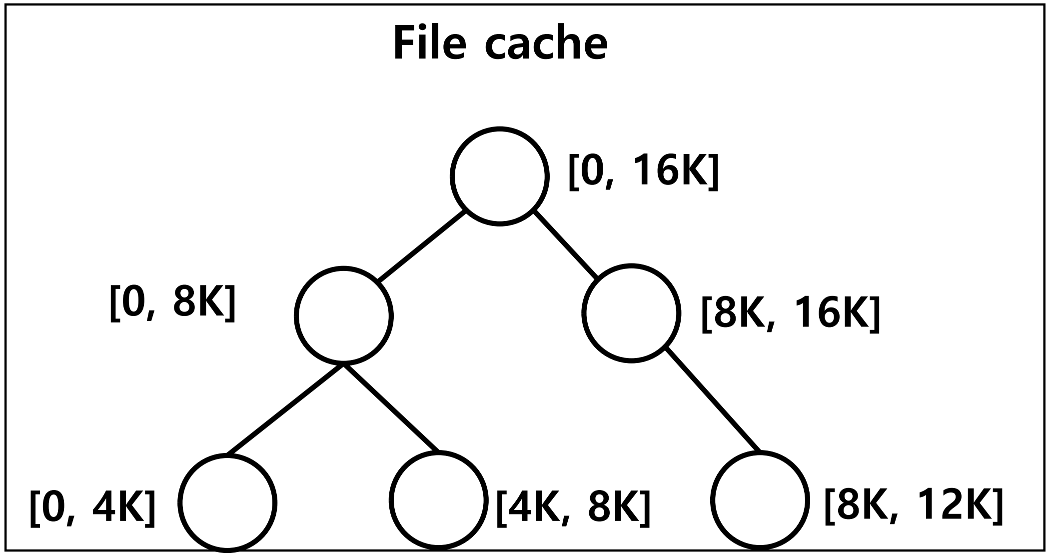


CrossPrefetch

- **Scalable and Concurrent Prefetching**
  - Using per-file range tree



xarray

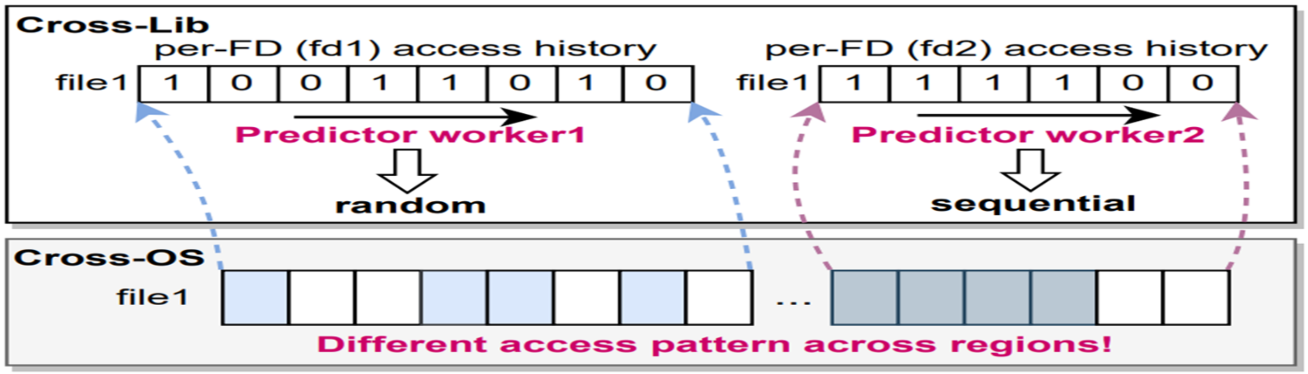


range tree

CrossPrefetch

Highly Random (000)	> 128KB distance
Random (001)	≤128KB distanc
Partially Random(010)	Mixed sequential and random access
Likely Sequential (011)	Sequential access with occasional random accesses
Stride-based Sequential (100)	Sequential access with fixed strides
Definitely Sequential(110)	Continuous sequential access

- **Low-overhead Prediction and Prefetching**
  - **Pattern detector**
    - Uses a simple n-bit counter for detecting a file's access pattern
  - **Support for File-descriptor Prefetching**
    - Maintain an access pattern detector for each file descriptor
    - The user-space file descriptor structure contains block range information and an access pattern counter



CrossPrefetch

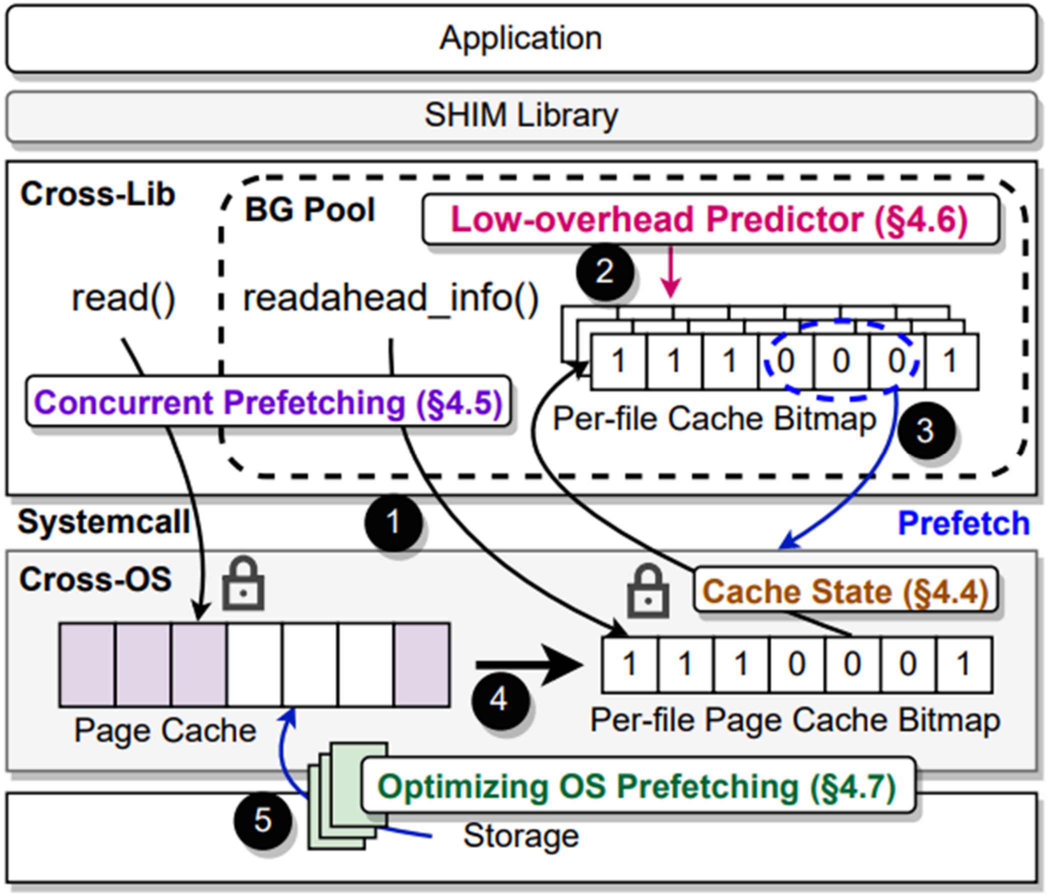
Highly Random (000)	> 128KB distance	No prefetching
Random (001)	≤128KB distanc	128KB
Partially Random(010)	Mixed sequential and random access	512KB~1MB
Likely Sequential (011)	Sequential access with occasional random accesses	2MB
Stride-based Sequential (100)	Sequential access with fixed strides	512KB~1MB
Definitely Sequential(110)	Continuous sequential access	2MB

- **Low-overhead Prediction and Prefetching**
  - **Memory-aware Aggressive Prefetching and Eviction**
    - Perform aggressive prefetching from the beginning of execution to reduce initial cache misses
    - Adjust prefetching aggressiveness by setting a High Threshold and a Low Threshold
    - Assume sequential access when a file is first opened
  - **Aggressive Reclamation**
    - Reclaim inactive file pages based on the LRU policy
    - Optimize cache usage for large files
  - **Support for Memory-Mapped I/O**
    - Utilize the OS-provided cache bitmap to detect access patterns
    - Analyze access patterns using background threads and apply an appropriate prefetching window size
    - However, this approach is similar to existing Linux OS prefetching and may have lower accuracy

CrossPrefetch

- **Optimizing OS Prefetching Path**
  - Linux limits the incremental prefetch threshold to 32 pages (128KB)
  - Extend the OS to dynamically increase the prefetch threshold using the info structure in the readahead_info system call
    - Restrict actual prefetch request sizes to a maximum of 64MB
      - ✓ Excessively large prefetch requests may degrade blocking I/O (read/write) performance

CrossPrefetch



Evaluation

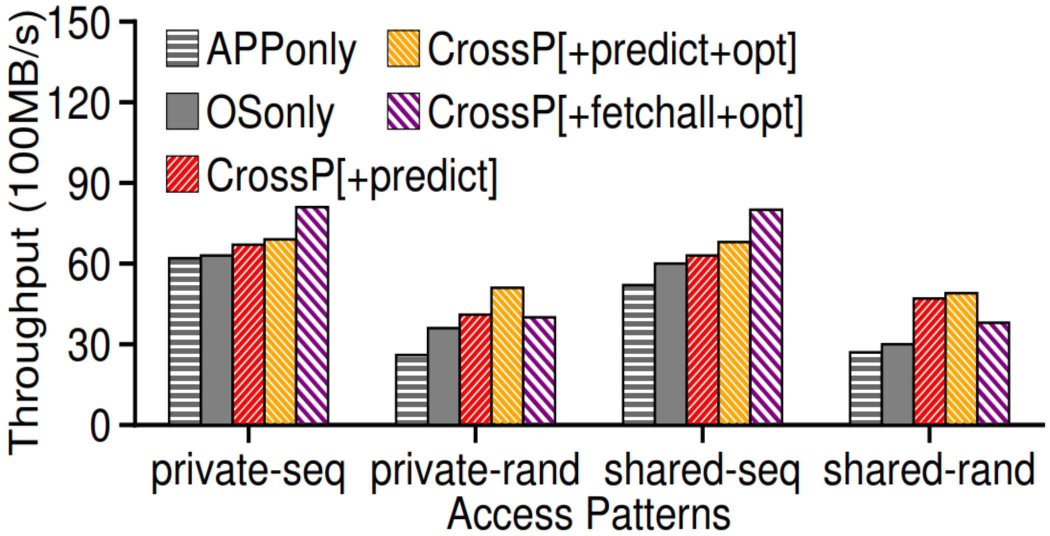
- **Experimental Environment**
  - **Configuration**
    - CPU: Intel Xeon Gold 6226R (16 cores, 2.9GHz)
    - Memory: 128GB DDR4
    - Storage: Samsung PM1733 NVMe SSD (6.4TB)
  - **Benchmark Tools**
    - fio, RocksDB, TPC-C (OLTP workloads)

Evaluation

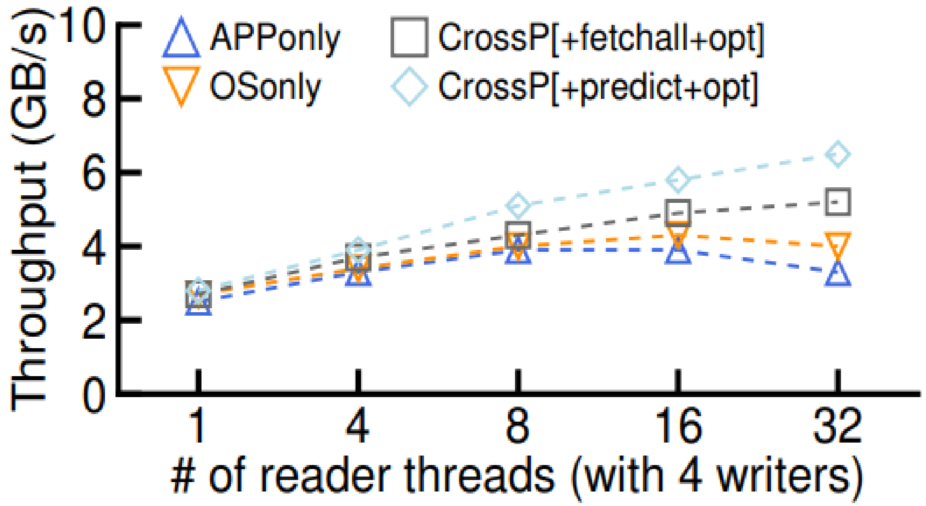
- **Microbench**
  - sequential & random access

Mechanism	Description
APPonly	Application tailored prefetching using readahead calls
OSonly	Prefetching delegated to OS and application prefetching is disabled
CrossP[+predict]	Fine-grained and low-interference prediction avoiding prefetching entire file
CrossP[+predict+opt]	*CrossP[+predict]* without OS limits but also provide memory-centric aggressive prefetching and eviction
CrossP[+fetchall+opt] (memory insensitive)	Proposed CrossPrefetch that uses cache state awareness to prefetch missing blocks of a file using `readahead_info()` assumes all data fits in memory

Table 2. Comparison Approaches



Microbench Read performance



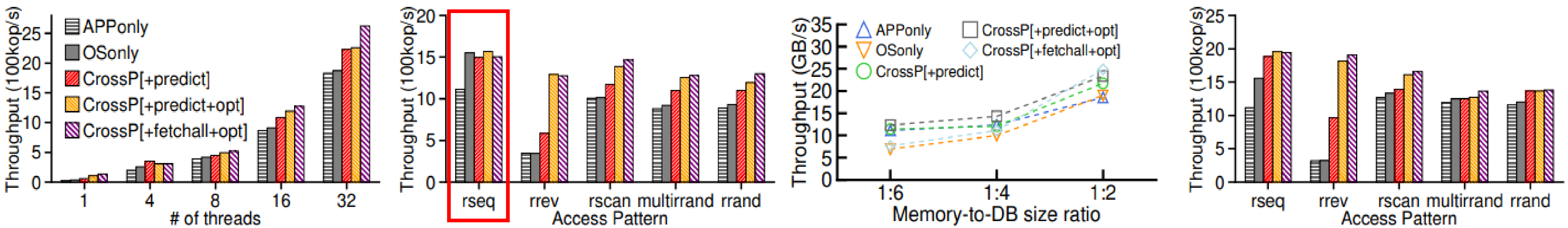
Microbench performance (4 writers, 4readers)

Evaluation

- **Local NVMe**
  - RocksDB dbbench

Mechanism	Description
APPonly	Application tailored prefetching using `readahead` calls
OSonly	Prefetching delegated to OS and application prefetching is disabled
CrossP[+predict]	Fine-grained and low-interference prediction avoiding prefetching entire file
CrossP[+predict+opt]	*CrossP[+predict]* without OS limits but also provide memory-centric aggressive prefetching and eviction
CrossP[+fetchall+opt] (memory insensitive)	Proposed CrossPrefetch that uses cache state awareness to prefetch missing blocks of a file using `readahead_info()` assumes all data fits in memory

Table 2. Comparison Approaches



(a) Multi-read Random on ext4 **(b) Access Patterns on ext4** **(c) Memory capacity impact** **(d) Access Patterns on F2FS**

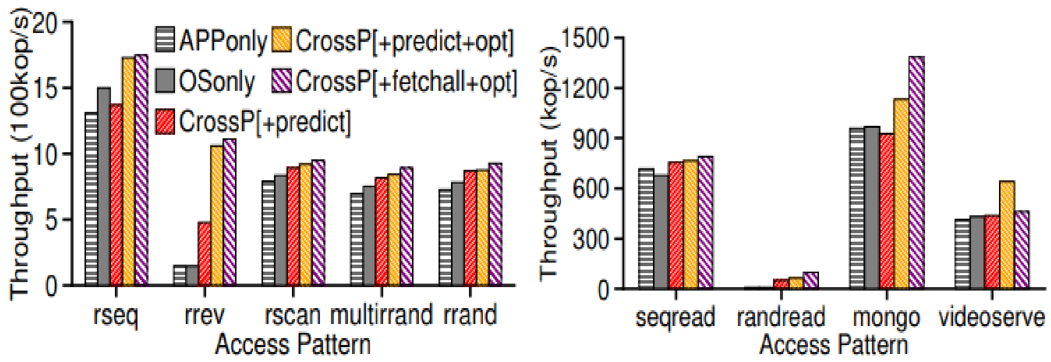
RocksDB DBbench on Local NVMe

Evaluation

- **Local & Remote NVMe**
  - RocksDB dbbench, Filebench, YCSB, Snappy

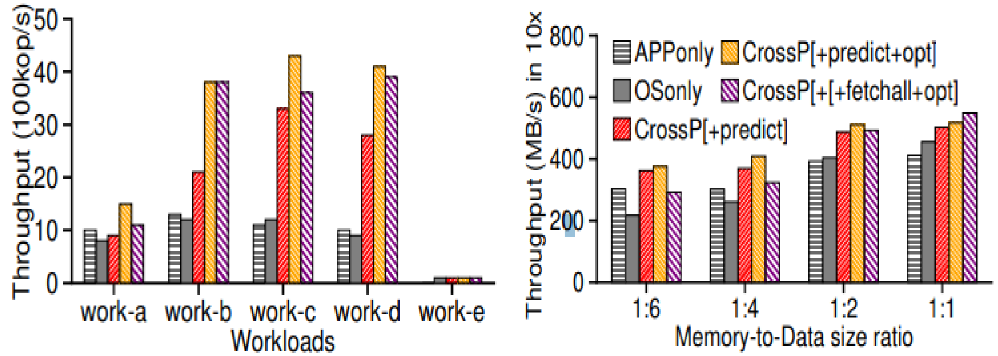
Mechanism	Description
APPonly	Application tailored prefetching using `readahead` calls
OSonly	Prefetching delegated to OS and application prefetching is disabled
CrossP[+predict]	Fine-grained and low-interference prediction avoiding prefetching entire file
CrossP[+predict+opt]	*CrossP[+predict]* without OS limits but also provide memory-centric aggressive prefetching and eviction
CrossP[+fetchall+opt] (memory insensitive)	Proposed CrossPrefetch that uses cache state awareness to prefetch missing blocks of a file using `readahead_info()` assumes all data fits in memory

Table 2. Comparison Approaches



(a) Access Patterns (b) Filebench Workloads

RocksDB on Remote NVMe (8a) and Filebench (8b)



(a) YCSB workload (b) Snappy

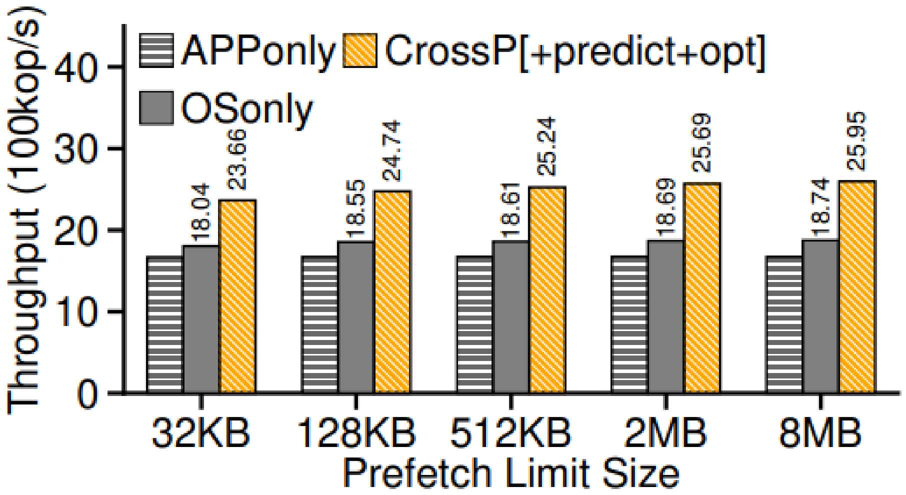
Real-world Workloads

Evaluation

- **Prefetch Limit Impact**

Mechanism	Description
APPonly	Application tailored prefetching using `readahead` calls
OSonly	Prefetching delegated to OS and application prefetching is disabled
CrossP[+predict]	Fine-grained and low-interference prediction avoiding prefetching entire file
CrossP[+predict+opt]	*CrossP[+predict]* without OS limits but also provide memory-centric aggressive prefetching and eviction
CrossP[+fetchall+opt] (memory insensitive)	Proposed CrossPrefetch that uses cache state awareness to prefetch missing blocks of a file using `readahead_info()` assumes all data fits in memory

Table 2. Comparison Approaches



Prefetch Limit Impact

Conclusion

- **Lack of communication between the OS and userspace**
  - The lack of direct communication between the OS and userspace in traditional prefetching mechanisms results in **inefficient cache management and unnecessary I/O operations**.
  - **Conflicting motivations and assumptions** between the OS and applications reduce overall system performance.

- **Harmony between layers in the system reduces friction**
  - **CrossPrefetch** enhances coordination between OS-level and userspace prefetching by **sharing per-file cache state and prefetching hints**.
  - **Sharing "Need-To-Know" information between layers significantly improves prefetch accuracy and system performance**.

Q&A

Q&A