Adaptive Prefetching for Accelerating Read and Write in NVM-based File Systems

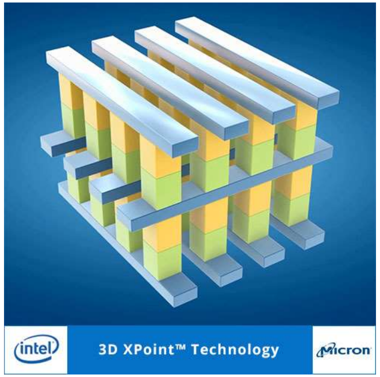
Shengan Zheng, Hong Mei, Linpeng Huang, Yanyan Shen, Yanmin Zhu

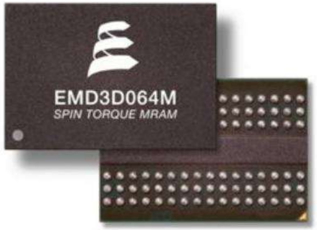
Department of Computer Science and Engineering
Shanghai Jiao Tong University

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

NVMM File Systems

- Non-Volatile Memory
  - √ Non-Volatile
  - √ Byte-addressable
  - × Longer latency than DRAM
  - × Lower bandwidth than DRAM
- NVMM file systems
  - SCMFS, BPFS, PMFS
  - NOVA, SIMFS, HiNFS
  - Not adaptive to different file access patterns





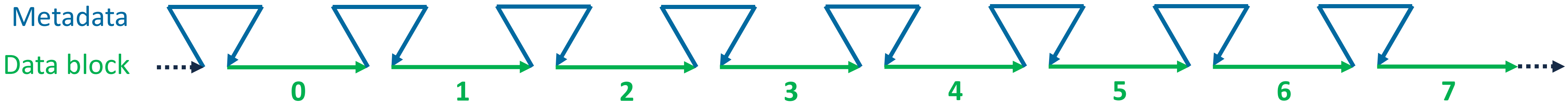
Motivation

- Higher performance
- Faster read & write

	Faster Read	Faster Write
Bottlenecks	Indirection of file inner structure	NVM write latency
Approaches	Continuous file address space	DRAM buffer
Proposed by	SIMFS (TOC '16)	HiNFS (EuroSys '16)

Motivation – Faster Read

- Bottleneck: locating pages with software routines
- **Continuous file address space**
- Normal access routine



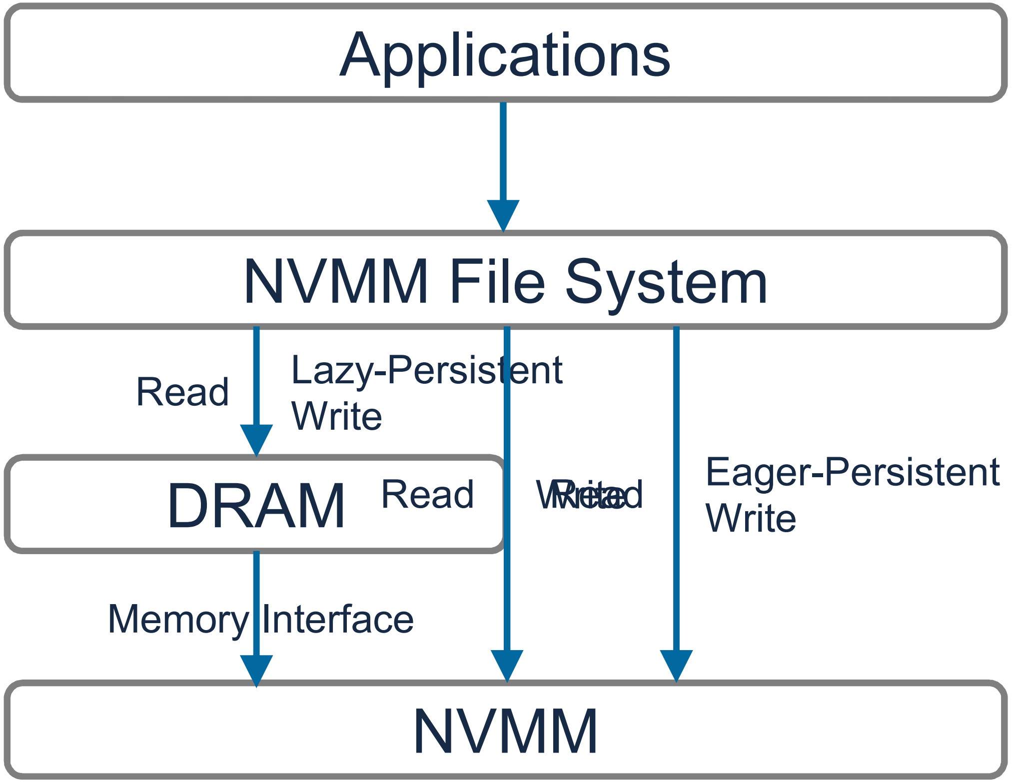
- Continuous file address space

- Not suitable for out-of-place writes

Motivation – Faster Write

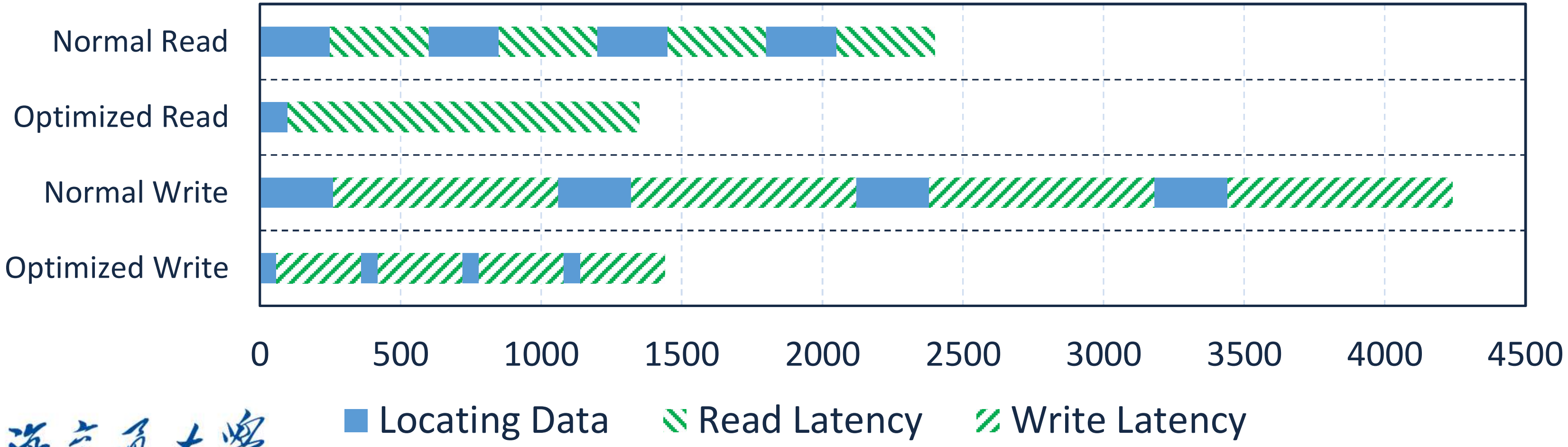
- Bottleneck: High write latency of NVM
- **DRAM write buffer**
- Perform write back on SYNC

- Introduces additional lookup overhead for read



Motivation – Merge

- NVMM File system with the best performance:
- Read with continuous file address space
- Write with DRAM write buffer

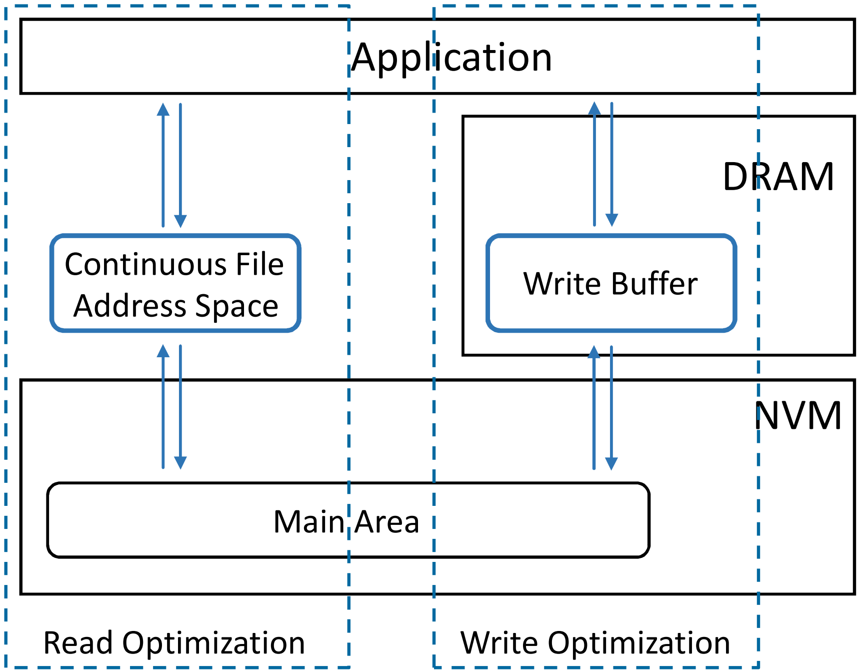


Motivation

	Faster Read	Faster Write
Bottlenecks	Indirection of file inner structure	NVM write latency
Approaches	Continuous file address space	DRAM buffer
Proposed by	SIMFS (TOC '16)	HiNFS (EuroSys '16)
Can we merge them into one NVM-based file system intuitively? No.		
Reasons	Not for out-of-place writes	Additional lookup overhead
	Optimized read **Slower write**	**Optimized write** **Slower read**

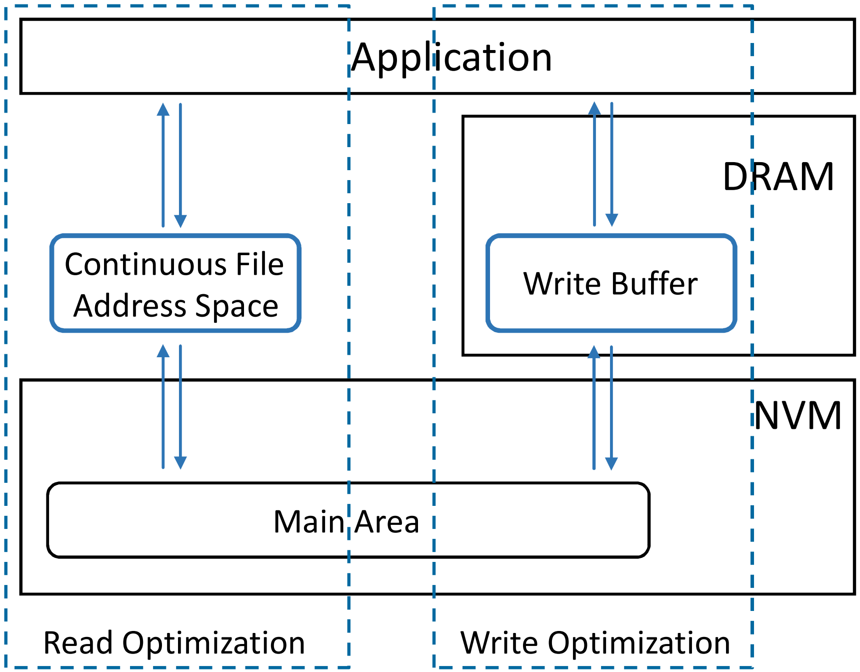
Goal

- Merge the read and write optimization approaches into one file system



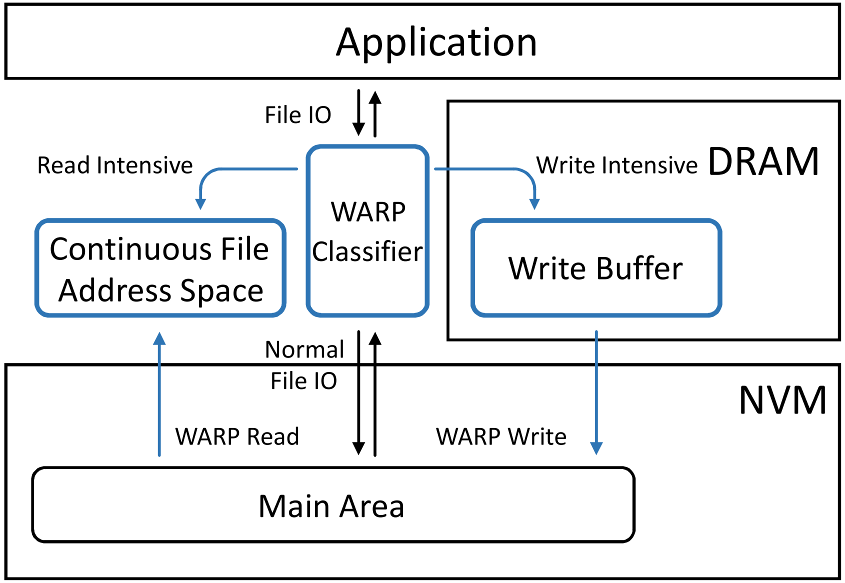
Challenges

- Merge the read and write optimization approaches into one file system
  - Adaptive optimization
  - Allocation overhead
  - Consistency



Design – WARP Classifier

- Classify read/write intensive accesses to files
  - Opened with *READ_ONLY* or *WRITE_ONLY* flag
  - Tagged with read/write-intensive by WARP benefit model
- Assign to different acceleration approaches accordingly

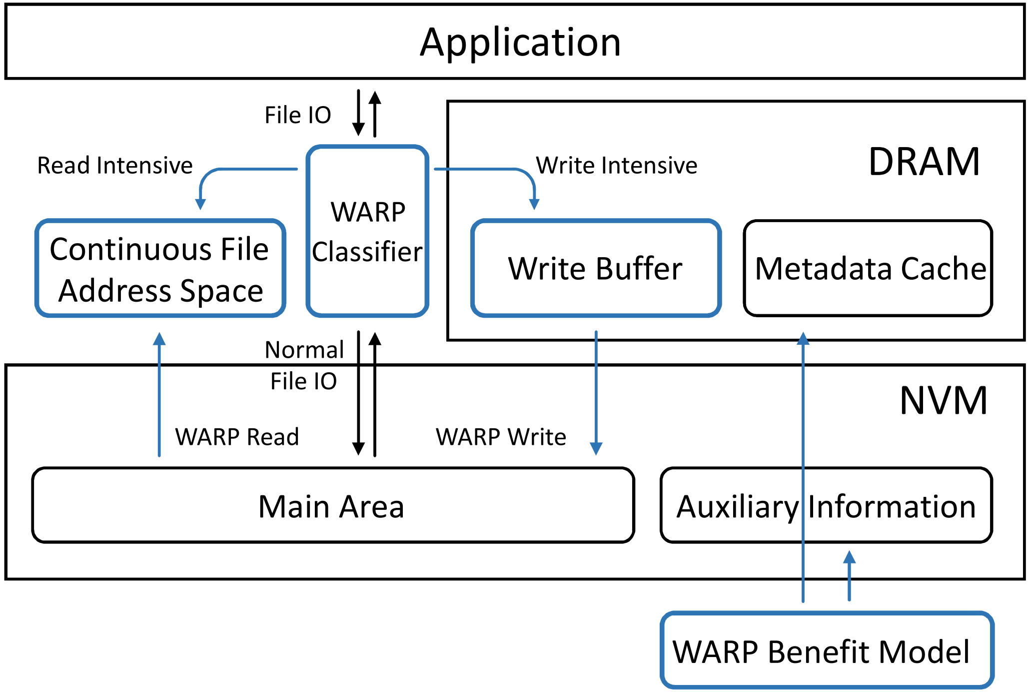


- *WARP: **W**rite **A**nd **R**ead **P**refetch*

Design – WARP benefit model

- To choose the best acceleration approach
- WARP benefit model
  - NVM characteristics
  - File access patterns
- Estimated access latency:

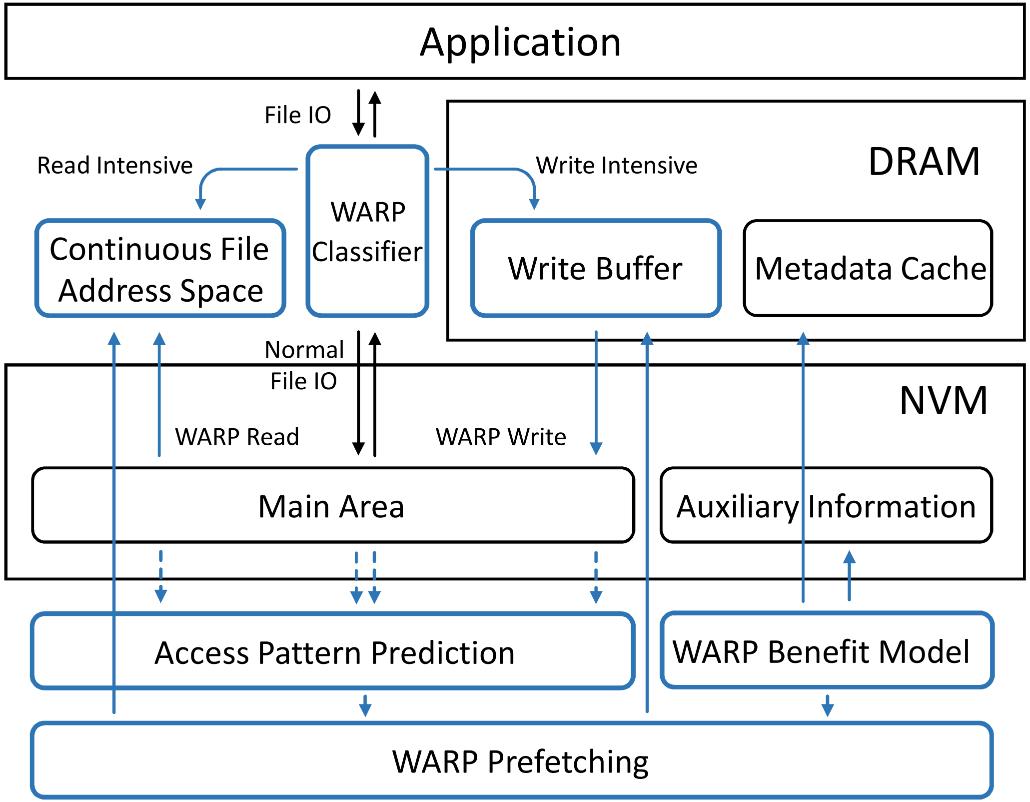
$$T = N_{Read} \times L_{ReadLatency} + S_{Read} \times V_{ReadBandwidth}^{-1} + N_{Write} \times L_{WriteLatency} + S_{Write} \times V_{WriteBandwidth}^{-1}$$



SHANGHAI JIAO TONG UNIVERSITY 上海交通大學

Design – Access Pattern Prediction

- Prefetching
  - Move the allocation steps out of critical path
  - Pre-allocation before files are accessed
- Collect file access traces and patterns
  - Which: successor prediction
  - How: access pattern prediction

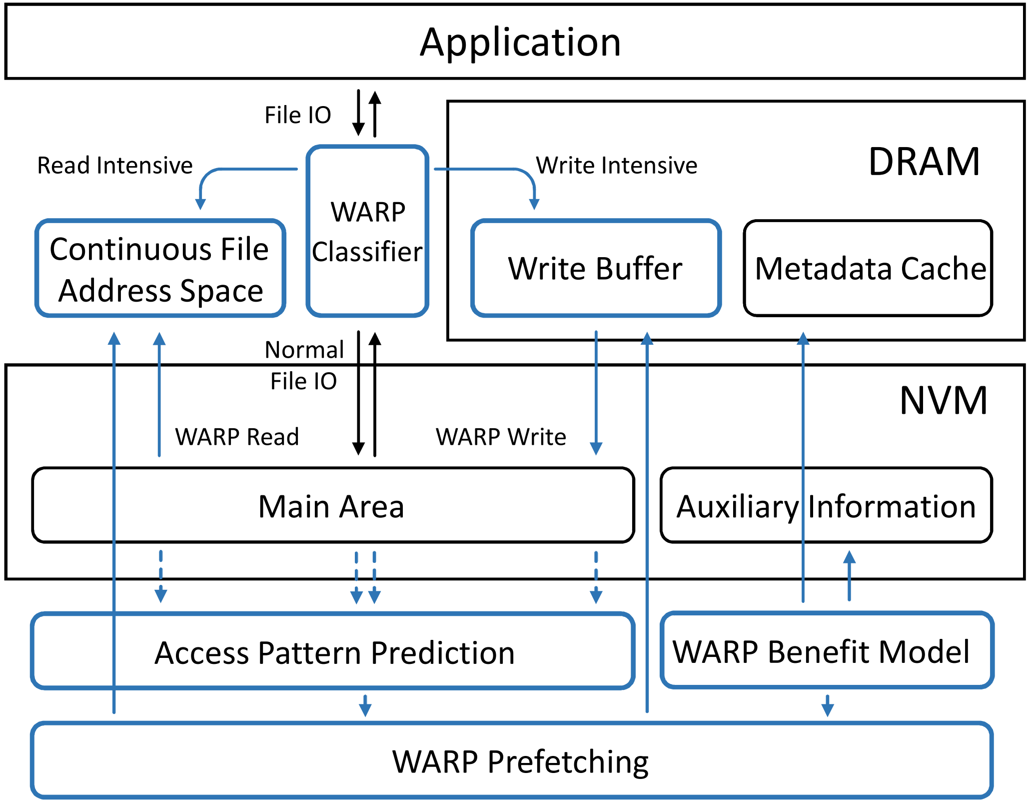


SHANGHAI JIAO TONG UNIVERSITY 上海交通大學

Design – WARP Prefetching

- WARP Prefetching
  - Whenever a file is accessed, prefetch the next.
  - File-based / process-based

- High overall performance
- High prefetch accuracy



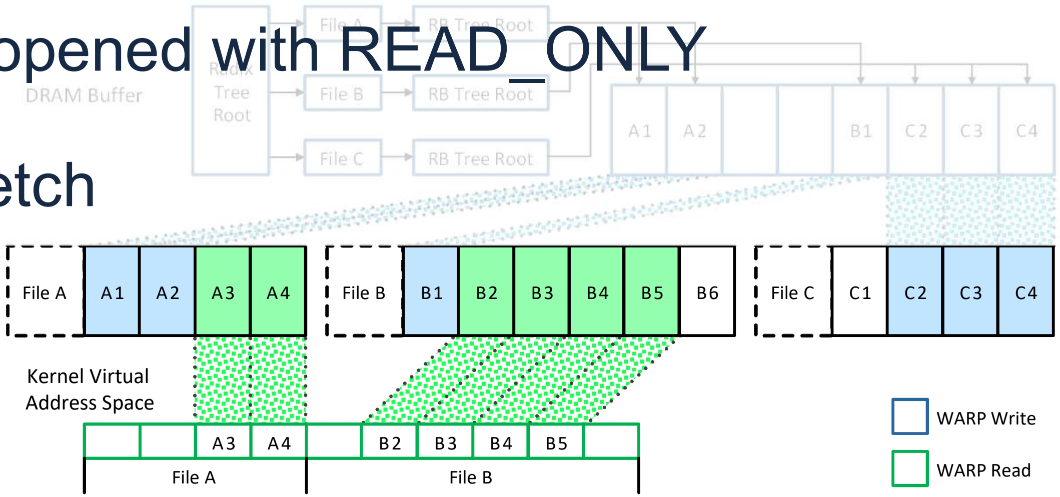
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Implementation – Granularity

- Granularity not too small (cacheline, block)
  - Adjacent data blocks share similar access patterns
  - The size of metadata will be enlarged
- Granularity not too big (file)
  - Optimization not precise
  - Additional optimization overhead
- We choose **2MB** as the granularity for our implementation

Implementation – Read Optimization

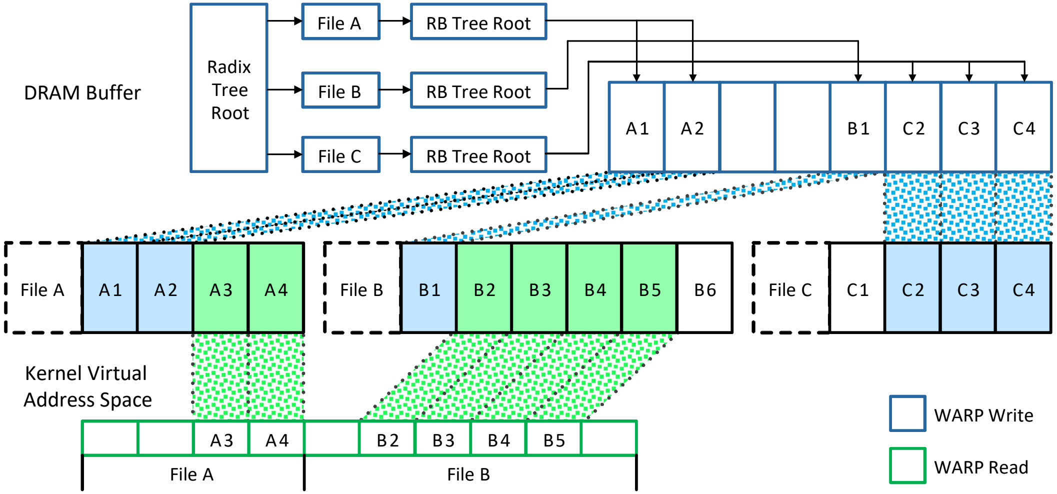
- Frequently-read nodes or files opened with READ_ONLY
- Background thread: warp_prefetch
  - Allocate virtual address space for the whole file
  - Map each valid block of the node



- After prefetching
  - Accessed directly and continuously through the page table entries via MMU
  - Handling out-of-place write: update the mapping address accordingly

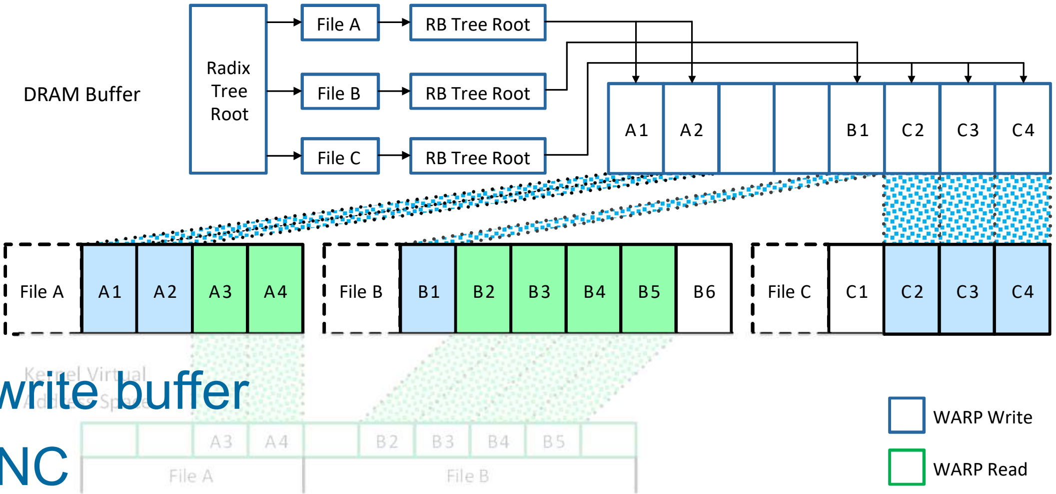
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Implementation – Read Optimization



Implementation – Write Optimization

- Frequently written nodes or files opened with WRITE_ONLY
- Background thread: warp_prefetch
  - Radix tree for files
  - Red-black tree for nodes

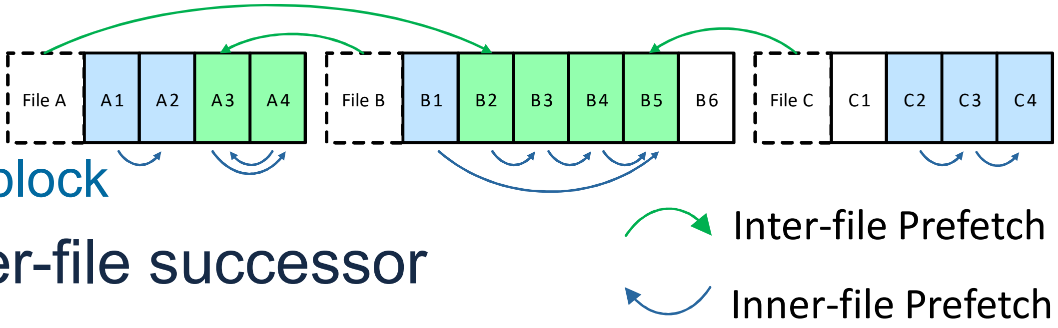


- After prefetching
  - Writes are intercepted by DRAM write buffer
  - Write back to NVM only when SYNC

Implementation – Successor Prediction

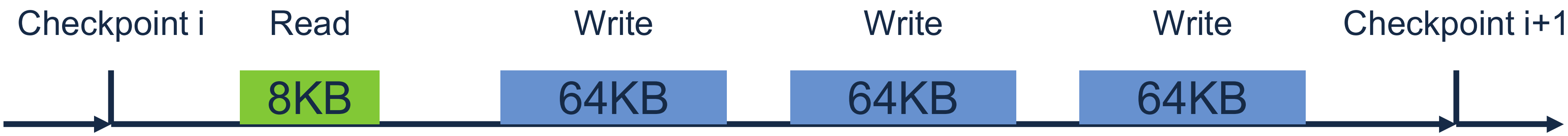
- Predict future node and file access

- Inner-file prediction
  - The **next node** within the file to be accessed
  - Stored in the metadata of **node** block
- Inter-file prediction
  - The **next file** to be accessed
  - Stored in the metadata of **inode** block
- Prefetch both inner-file and inter-file successor



Implementation – WARP benefit model

- Objective: minimizing the node's overall I/O time between two consecutive checkpoints

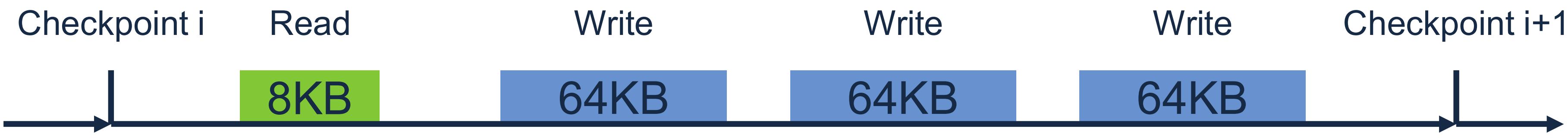


- $T=2*(250+50)+8KB*(8GB/s)^{-1}+48*(250+500)+192KB*(2GB/s)^{-1}=134\mu s$
- T = [Read overhead] + [Write overhead] + [*Writeback overhead]
- $T = N_{Read} \times L_{ReadLatency} + S_{Read} \times V_{ReadBandwidth}^{-1} + N_{Write} \times L_{WriteLatency} + S_{Write} \times V_{WriteBandwidth}^{-1}$
- N: The number of access times   L: The access latency of file inner structure and memory
- S: The total size of the I/O access   V : The transmission bandwidth of memory

Implementation – WARP benefit model

- T(Read Opt.)

$$= N_{Read} \times L_{ReadLatency} + S_{Read} \times V_{ReadBandwidth}^{-1} + N_{Write} \times L_{WriteLatency} + S_{Write} \times V_{WriteBandwidth}^{-1}$$

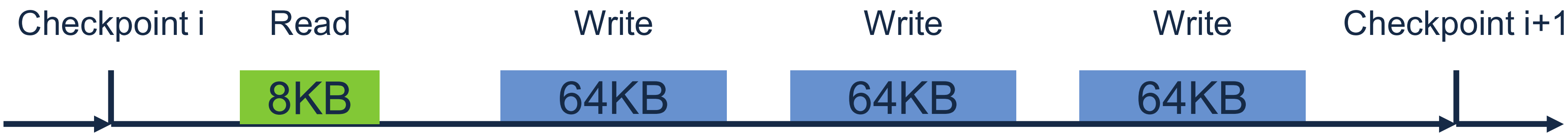


- T=1*(100+50)+8KB*(8GB/s)$^{-1}$+48*(250+500)+192KB*(2GB/s)$^{-1}$=133μs
- Continuous file inner structure reduces the access latency

Implementation – WARP benefit model

- T(Write Opt.)

$$= N_{Read} \times L_{ReadLatency} + S_{Read} \times V^{-1}_{ReadBandwidth} + N_{Write} \times L_{WriteLatency} + S_{Write} \times V^{-1}_{WriteBandwidth} + T_{WB}$$



- T=2*(250+15)+8KB*(15GB/s)$^{-1}$+48*(250+15)+192KB*(10GB/s)$^{-1}$

  \+ 16*(250+500)+64KB*(2GB/s)$^{-1}$ =77μs

- Writes to DRAM instead of NVM
- Write back to NVM on SYNC

Implementation – WARP benefit model

Checkpoint i | Read | Write | Write | Write | Checkpoint i+1

8KB | 64KB | 64KB | 64KB

T = 134μs   T = 133μs (↗0%)   T = 77μs (↗**74%**)

Checkpoint i | Read | Read | Read | Write | Checkpoint i+1

64KB | 64KB | 64KB | 8KB

T = 28μs   T = 14μs (↗**100%**)   T = 24μs (↗17%)

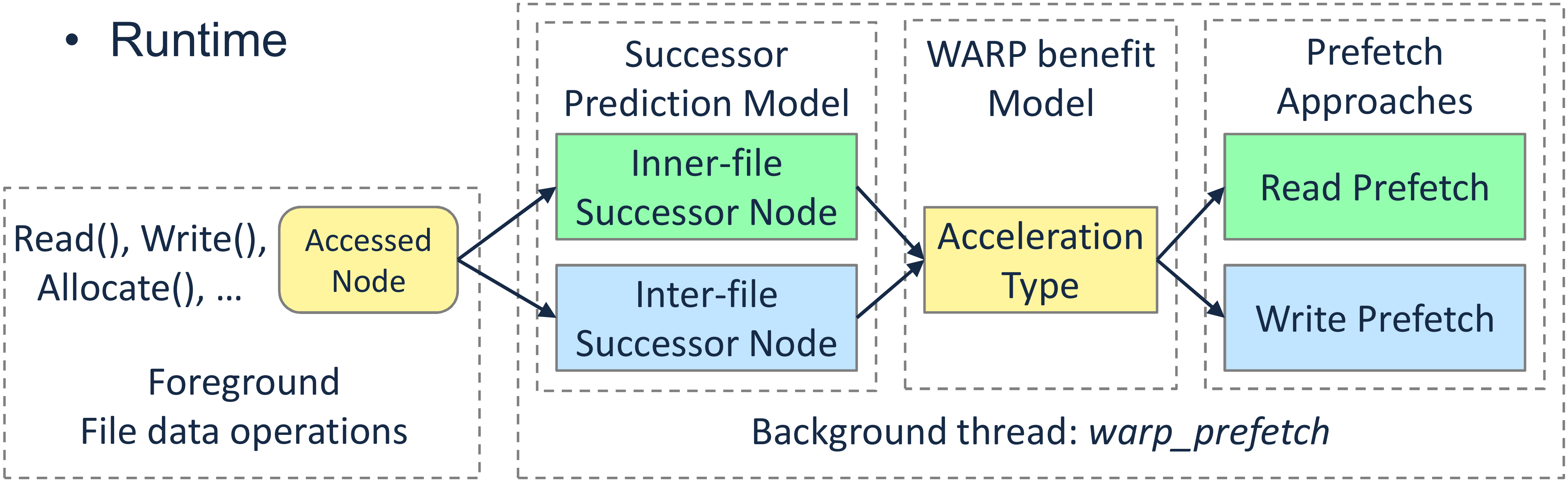
Checkpoint i | Read | Write | Read | Write | Read | Write | Checkpoint i+1

16KB | 16KB | 16KB | 16KB | 16KB | 16KB

T = 39μs   T = 35μs (↗9%)   T = 23μs (↗**66%**)

Implementation – Prefetching

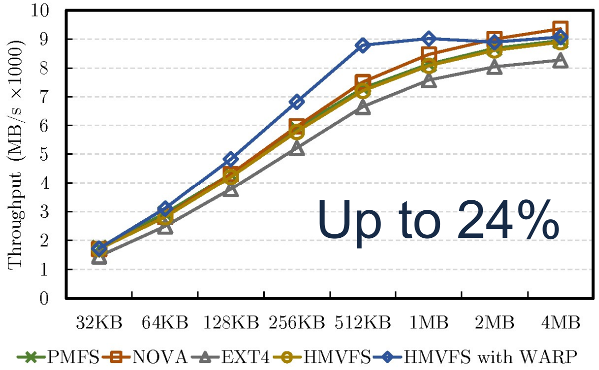
- Runtime

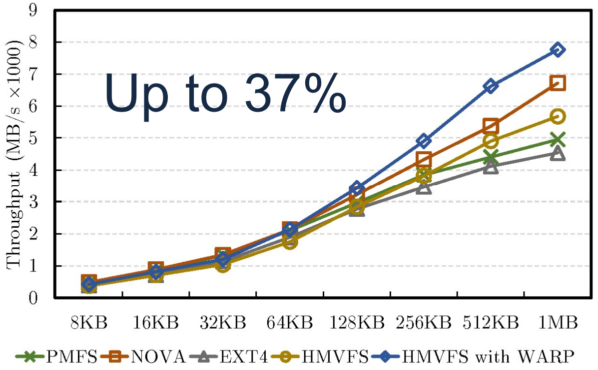


Evaluation – Micro-benchmarks

- Experimental Setup
  - A commodity server with 64 Intel Xeon 2GHz processors and 128GB DRAM
  - Latency: Read 50ns Write 500ns
  - Bandwidth limitation: Read 10GB/s Write 2GB/s

File Read

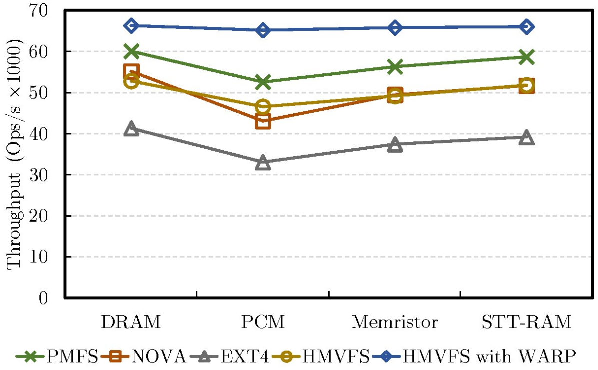




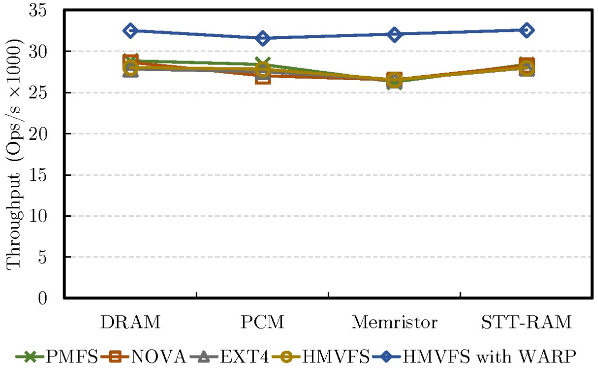
File Write

Evaluation – Macro-benchmarks

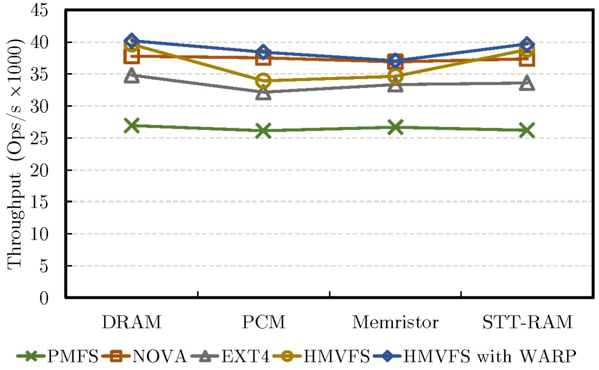
Fileserver



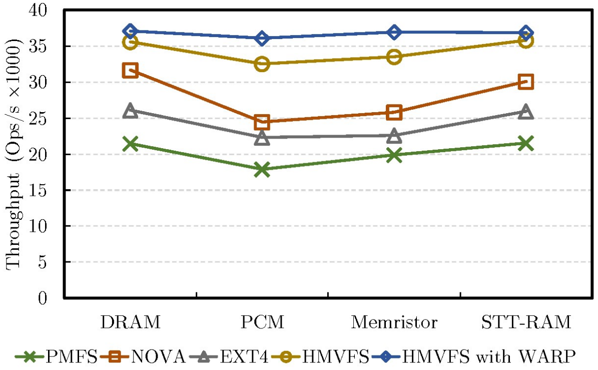
Webserver



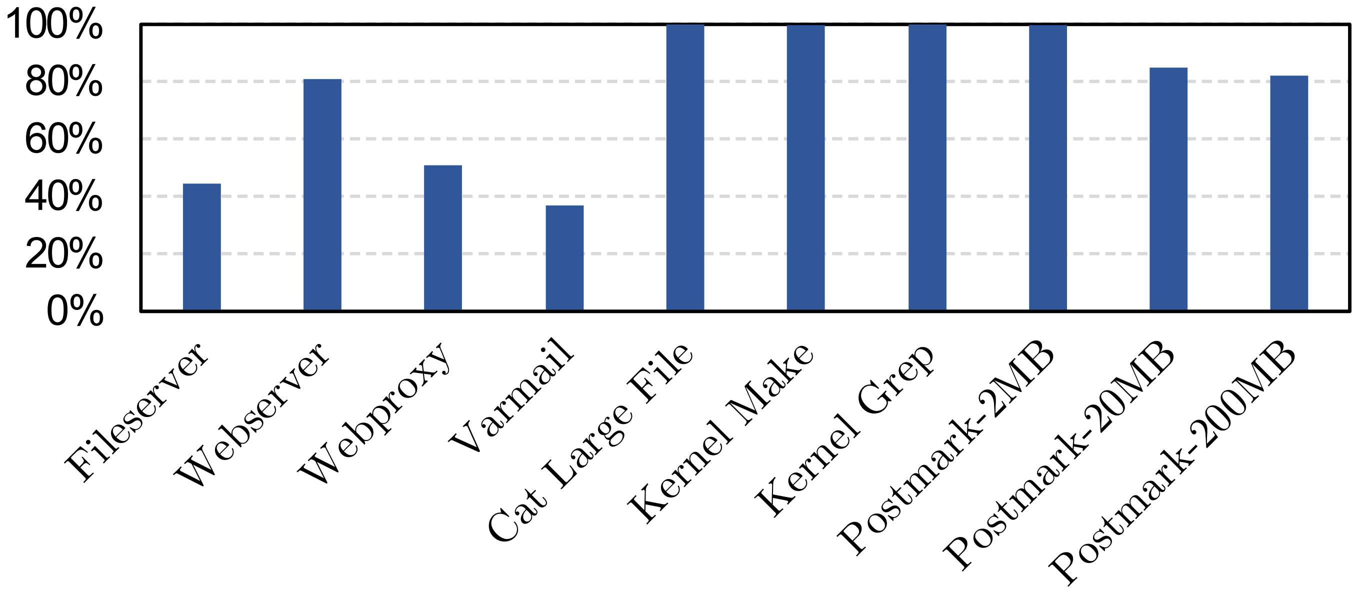
Webproxy



Varmail



Evaluation – Prefetching accuracy



Prefetching accuracy

Conclusion

- Design adaptive prefetching strategy to deploy read/write optimization approach
- Implement file access successor prediction model to predict future file access pattern
- Implement WARP benefit model to calculate the most beneficial optimization approach
- Efficient prefetching for NVM-based file systems

Thanks!