Transforming Legacy File Systems into Persistent Memory Exploiting File Systems with MeLo@V

Hyunsub Song, Young Je Moon, Se Kwon Lee and Sam H. Noh

UNIST
(Ulsan National Institute of Science and Technology)

NECSST Next-generation Embedded / Computer System Software Technology

Motivation

- **The advent of Persistent Memory (PM)**
- **PM storage targeted file systems to date**
  - Premise is that legacy file systems are sub-optimal
- **However, legacy file systems are time tested**
  - Many years of time and effort ingrained
  - Matured with time

NECSST Next-generation Embedded / Computer System Software Technology

Question raised

- Can we not make use of the mature features of legacy file systems, while, at the same time, reaping the high performance offered by PM?
- To answer this question:
  we first go through a thorough analysis of legacy file systems
  - Fine-grained I/O flow measurement for various file systems

Empirical analysis

- **Measurement for traditional file system I/O overhead (us)**

Component		Description	EXT4		DAX	PMFS	NOVA
			Async	Sync			
System Call	System call gate	Internal system call function	0.3	0.3	0.2	0.3	0.3
VFS Layer	VFS objects	Set structure related to VFS	1.0	1.0	0.9	0.9	0.8
	I/O type switch	Change type of I/O	3.2	5.7	2.2		
FS specific Layer	Page cache	Work related to page cache	17.3	16.8			
	Memory I/O	Write data to memory	0.4	0.5			
	Page cache flush	Flush dirty page to storage		33.1			
	FS consistency	Mechanism for FS consistency		101.1	7.1		
	DAX I PMFS I NOVA	Write data to storage			13.2	19.1	19.3
Total Elapsed Time			22.2	158.5	23.6	20.3	20.4

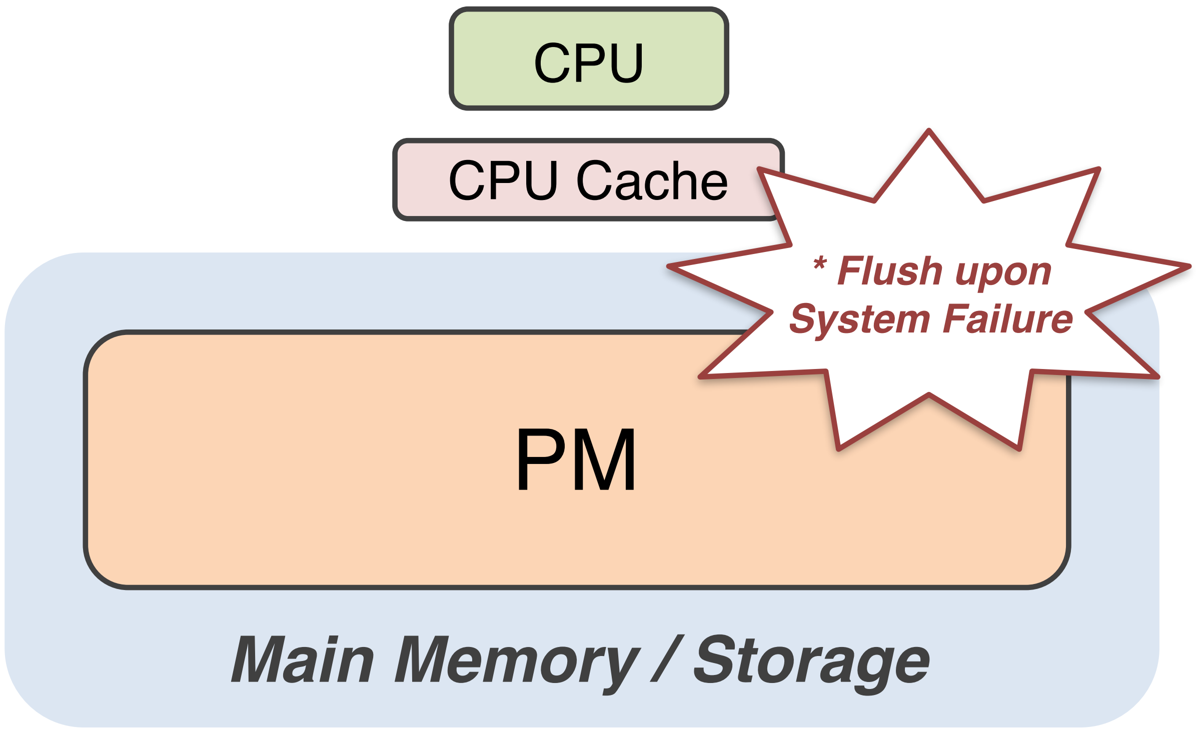
Most of the overhead at this point

Our Answer to Question Raised

- Yes, it is possible to achieve high performance with legacy file systems!
- How?
  - Remove synchronous flush of data in page cache
  - Optimize file system consistency mechanism

Our Assumption

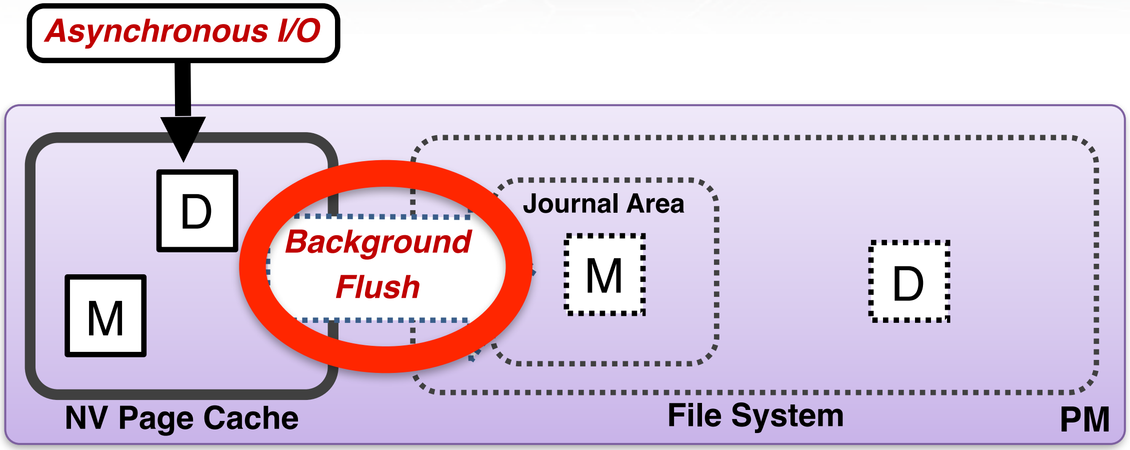
- **PM-only system**



Data in <u>main memory</u> is retained after a system crash!!!

* "Whole-System Persistence", In Proc. ASPLOS, 2012.

Through assuming PM-only system



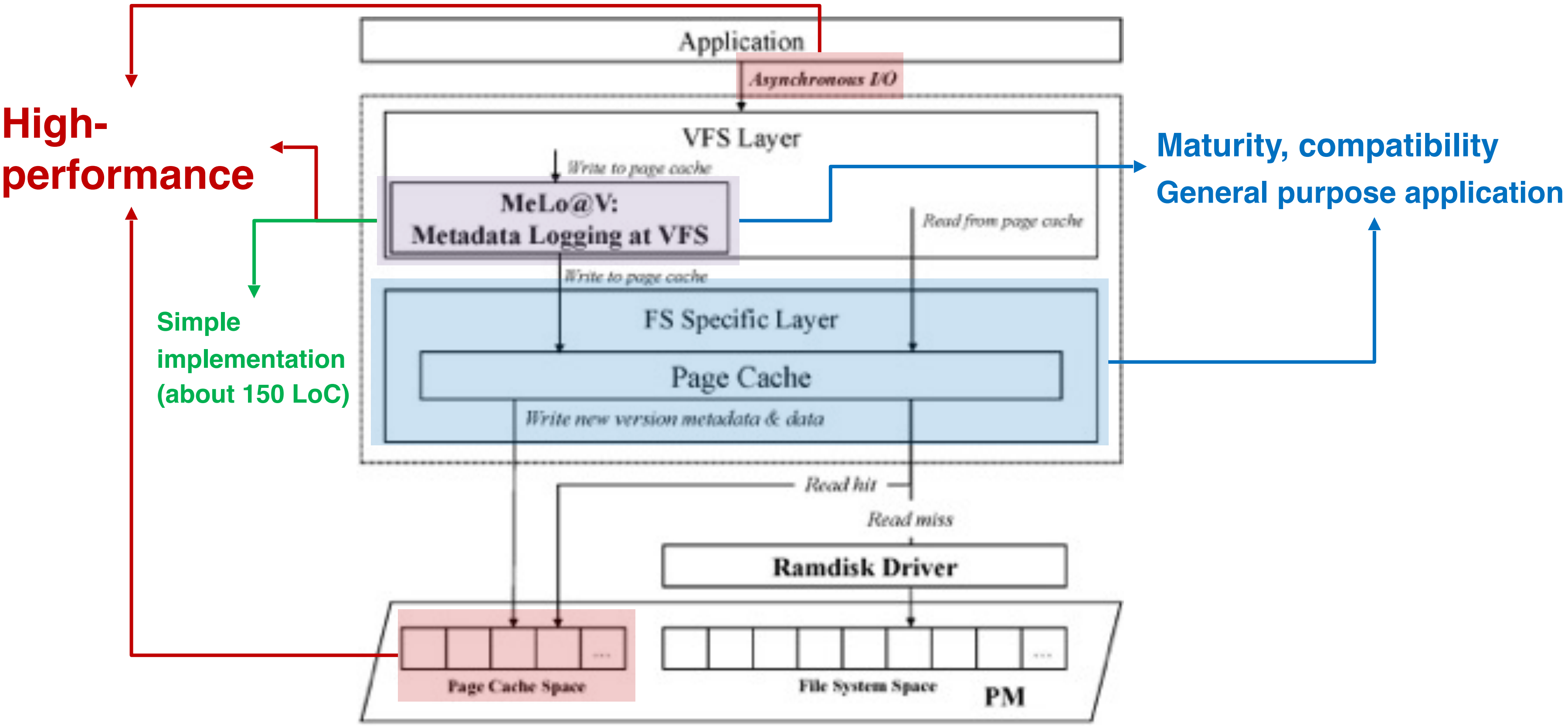
- **We can use**
  - Asynchronous I/O
    - for hiding the overhead of 'page cache flush' component
    - for hiding the overhead of 'FS consistency' component
  - Natural multi-versioning structure (use page cache area as multi-versioning area)
- **However, when in-place-update occurs in NV Page Cache**

  >> Need to guarantee sanity of data and metadata

  >> MeLo@V: a lightweight logging mechanism

Metadata Logging at the VFS Layer (MeLo@V)

- MeLo@V can
  - Guarantee file system consistency using lightweight logging mechanism
  - Translate legacy file system into PM exploiting file system



MeLo@V integration into legacy file systems

Logging in MeLo@V

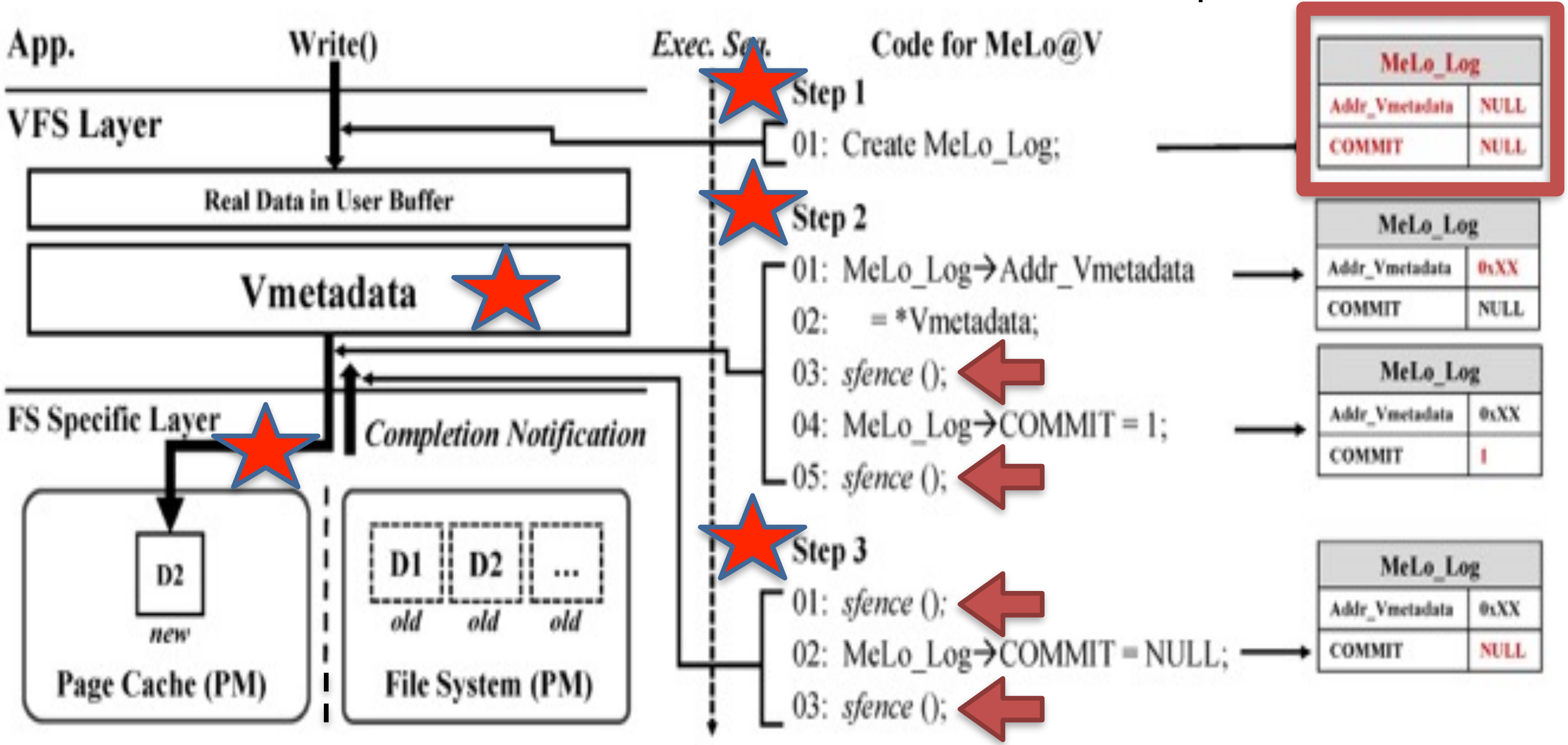
- **In DIMM based PM storage**
  - **For guaranteeing durability of data**

    —> *~~CLFLUSH~~* **Flush on failure through *WBINVD* instruction**
  - **For prevention reordering of memory operations**

    —> *SFENCE*

MeLo@V

- **Logging in MeLo@V**
  - MeLo@V code inserted in traditional write execution sequence



Our target system is based on Linux platform.
The Vmetadata that we define are *kiocb* and *iovec* structures in the Linux kernel.

Recovery in MeLo@V

MeLo@V

- **Recovery in MeLo@V**
  - Consider COMMIT marker value in MeLo_Log upon reboot

Case 1

MeLo_Log	
Addr_Vmetadata	NULL
COMMIT	NULL

COMMIT is NULL upon reboot
➔ MeLo_Log was created but never used
Action required: Remove MeLo_Log

MeLo_Log	
Addr_Vmetadata	0xXX
COMMIT	NULL

Case 2

MeLo_Log	
Addr_Vmetadata	0xXX
COMMIT	1

COMMIT is 1 upon reboot
➔ New write did not commit
Action required:

Replay the previous failed write using data obtained through Addr_Vmetadata (in Linux, ***file->f_op->write_iter(kiocb, iovec)*** is called)

Case 3

MeLo_Log	
Addr_Vmetadata	0xXX
COMMIT	NULL

COMMIT is NULL upon reboot
➔ New write was committed
Action required: Nothing
(page cache daemon will take care of the rest)

Evaluation

- **Experimental environment**

System configuration

	Description
CPU	Intel i7-4820K 3.7GHz (4 cores / 8 threads)
Memory	Samsung DDR3 8GB PC3-12800 X 8 (64GB)
OS	Linux CentOS 6.6 (64bit) kernel v4.3
PM storage	Emulated with Ramdisk (56GB)

Comparison group

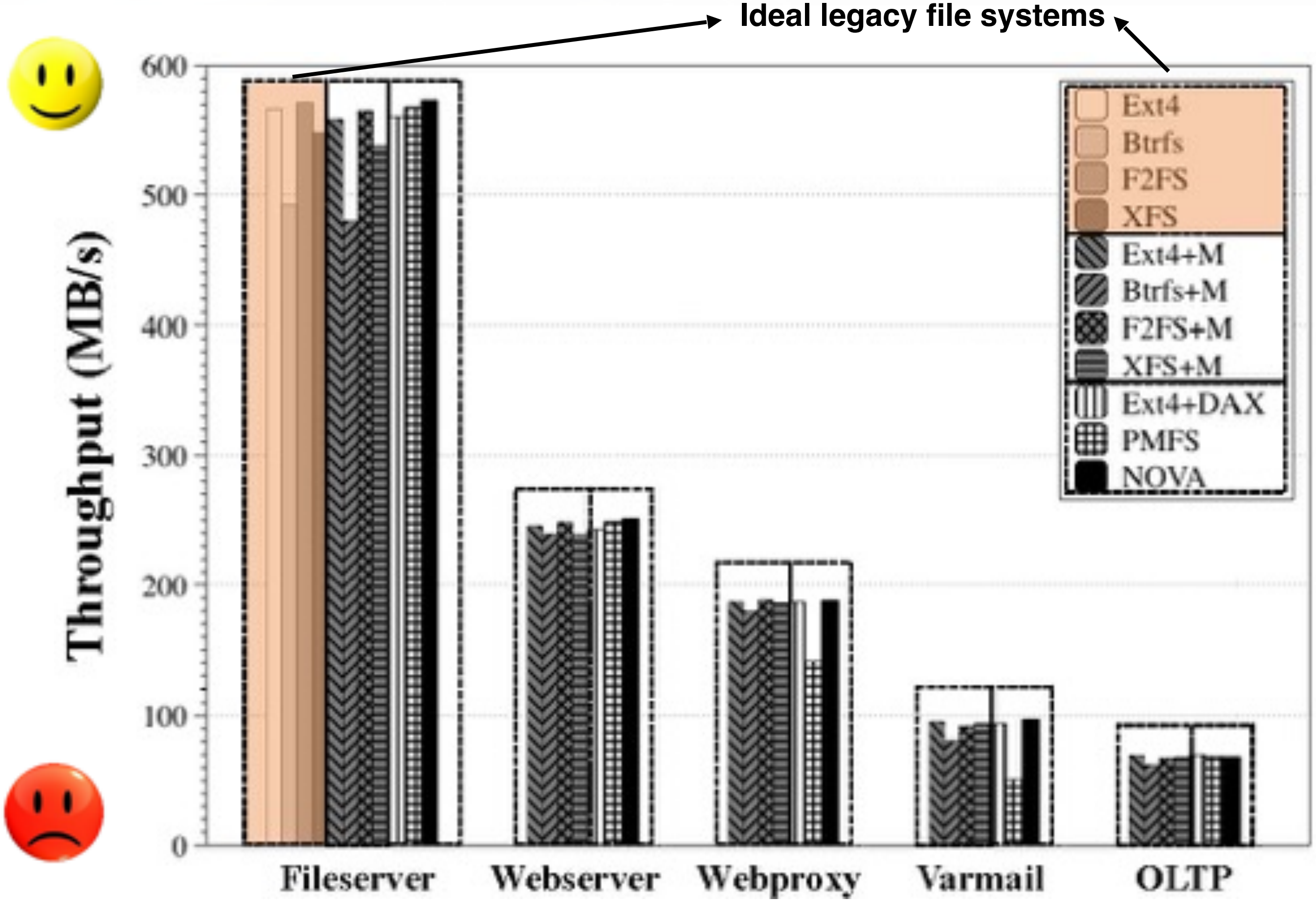
	Description
Ideal legacy file system	Original legacy file systems => Asynchronous I/O mode (Ext4, Btrfs, F2FS, XFS)
Legacy file system + M	Legacy file system + MeLo@V
Ext4+DAX	Ext4 with DAX
PMFS	PM-dedicated file system
NOVA	PM-dedicated file system

Characteristics of benchmarks

Filebench (10-15GB footprint)	R:W	Mean file size	# of files	# of threads
Fileserver	1:2	128K	100K	50
Webserver	10:1	32K	500K	50
Webproxy	5:1	32K	400K	50
Varmail	1:1	16K	800K	50
OLTP	1:1	1.5G	10	W: 10 R: 200
Key-value store	**R:W**	**Record selection**	**Dataset size**	**# of threads**
YCSB-A	1:1	Zipfian	10G	5
ForestDB	2:1	Zipfian	15G	5

Evaluation

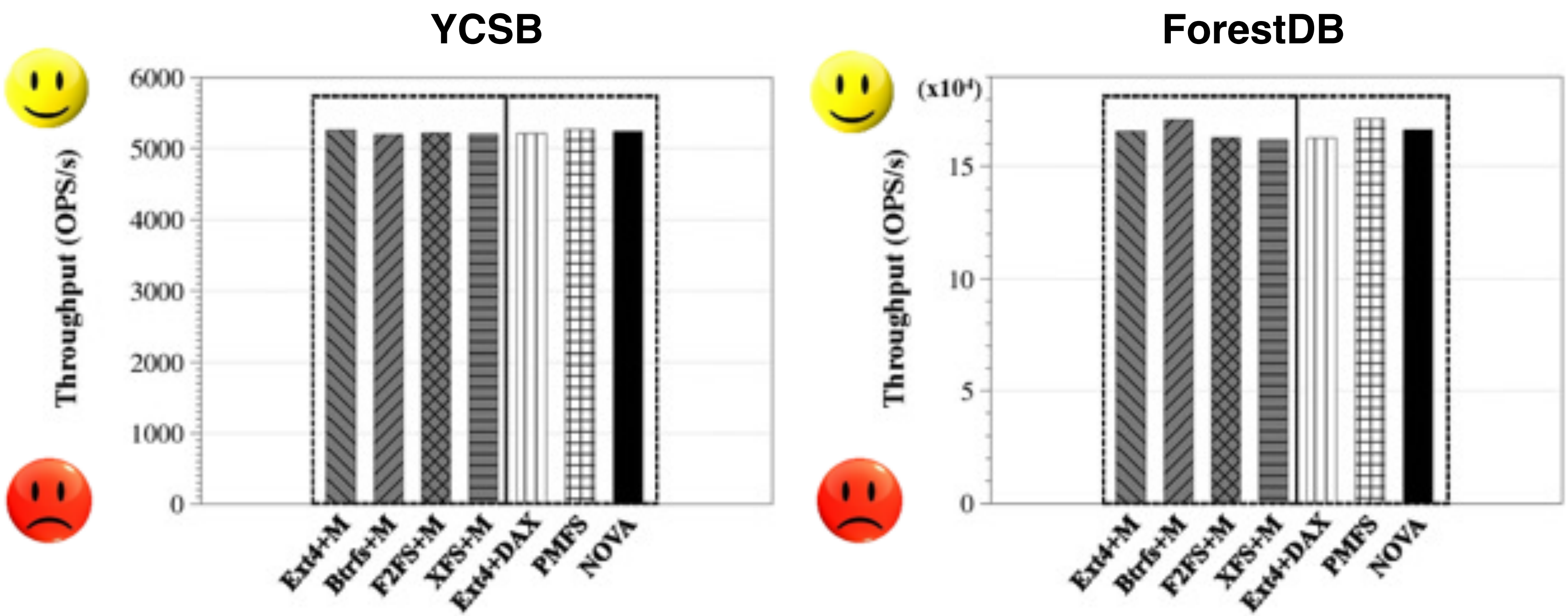
- **Overall performance** (Filebench)



Performance of Filebench benchmarks

* Even with Melo@V deployed, performance is only slightly below those of the ideal legacy file systems

- **Overall performance** (Key-value store)

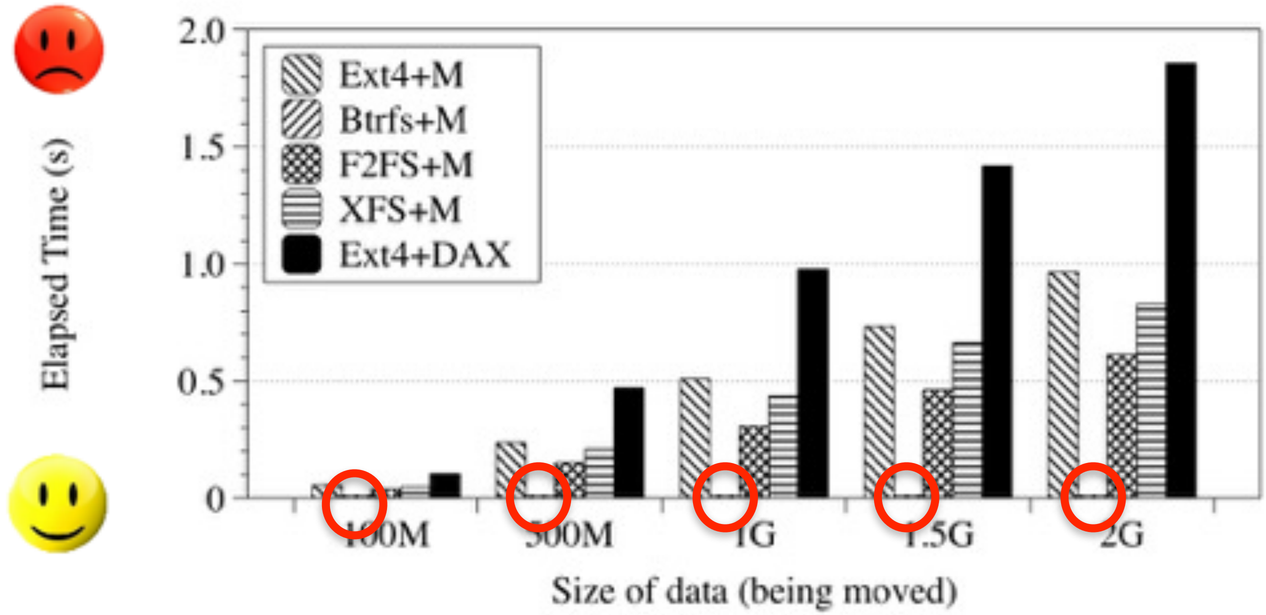


Performance of key-value store benchmarks

* **Legacy file systems transformed into PM exploiting ones through MeLo@V performs comparably to state-of-the-art PM-based file systems**

Evaluation

- **Page cache as the centerpiece**
  - We measure the elapsed time of *copy_file_range()* system call
  - This call optimizes performance of data passing between files a single mode change by making use of page cache



Function copy_file_range() performance as data size varied

* Making use of the page cache through MeLo@V performs considerably better than Ext4+DAX

* Btrfs+U spends essentially no time performing this system call for all data copy, because Btrfs uses *reflink* for data copy

  * Note that such peculiarities of file systems are naturally exploited with our MeLo@V approach

Conclusions

- **MeLo@V**
  - Retains traits of legacy file systems
  - Performs in par with state-of-the-art PM-based file systems
  - Can apply to various legacy file systems using VFS layer
  - Is implemented in roughly 150 lines of code

Thank you!!!

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY 2009

* E-mail: hssong1987@unist.ac.kr