Caching or Not: Rethinking Virtual File System for Non-Volatile Main Memory

Ying Wang, Dejun Jiang, Jin Xiong
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
Outline

• Background & Motivation
• Design & Implementation
• Evaluation
• Related Work
• Summary
Non-Volatile Memory

- Non-volatile memories (NVM)
  - PCM, ReRAM, STT-RAM, 3D xPoint technology
- NVM based file systems

Features of NVM\(^{[1, 2, 3, 4, 5]}\)

<table>
<thead>
<tr>
<th></th>
<th>NAND Flash</th>
<th>PCM</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte-addressable</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Density</td>
<td>4X</td>
<td>2X-4X</td>
<td>1X</td>
</tr>
<tr>
<td>Read latency</td>
<td>25us</td>
<td>50~70ns</td>
<td>60ns</td>
</tr>
<tr>
<td>Write latency</td>
<td>350us</td>
<td>150~1000ns</td>
<td>60ns</td>
</tr>
<tr>
<td>Write Endurance</td>
<td>(10^5)</td>
<td>(10^9)</td>
<td>(10^{16})</td>
</tr>
<tr>
<td>Non-Volatile</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

---

\(^{[1]}\) Dulloor, EuroSys’ 2014  
\(^{[2]}\) Yang, FAST’ 2015  
\(^{[3]}\) Lee, ISCA’ 2009  
\(^{[4]}\) Chang, ISSCC’ 2014  
\(^{[5]}\) Kannan, FAST’ 2018
Stack of Existing NVM File Systems

- Existing kernel-level NVM file systems
  - Retain virtual file system

Kernel-level file system
- BPFS\(^1\)
- SCMFS\(^2\)
- PMFS\(^3\)
- NOVA\(^4\)

\(^1\) Condit, SIGOPS 2009
\(^2\) Wu, SC 2011
\(^3\) Dulloor, EuroSys 2014
\(^4\) Haris, EuroSys 2014
Virtual File System (VFS)

• A software abstraction layer
  • First introduced to allow accessing local file system and remote Network File System transparently

• Current features
  • Supporting multiple file systems
  • Metadata caching
  • Concurrency control
  • Permission check
Small Files and Metadata Access in File System

• Small files in file systems
  • Desktop file system: more than 80% of accesses are to files smaller than 32B[1]
  • Cloud and HPC cluster: 25%~40% files < 4KB[2]

• Metadata operations dominate small file access
  • Metadata access contributes 40% time for accessing a small file on a disk [1]
  • For iBench system, 10%~20% of system calls do path lookup[4]
  • Path-based system calls account for 6%~54% of total execution time[5]

For NVM based file system, how about metadata operation overhead?

[1] Zhang, FAST 2016
[5] Tsai, SOSP 2015
Metadata Operation Overhead in NVM FS

• Path lookup overhead

VFS: time of lookup spent in VFS
FS: time of lookup spent in physical file system
Metadata Operation Overhead in NVM FS

• Path lookup overhead
  • From disk to NVM, total execution time is reduced
  • Execution times reduce by 28.4% ~ 73.4%

VFS: time of lookup spent in VFS
FS: time of lookup spent in physical file system
Metadata Operation Overhead in NVM FS

• Path lookup overhead
  • From disk to NVM, total execution time is reduced
    • Execution times reduce by 28.4% ~ 73.4%
  • From disk to NVM, proportion of path lookup operations increase
    • Percentages of execution times in VFS increase by 16.5%~459.6%

VFS brings extra overhead for NVM based FS

VFS: time of lookup spent in VFS
FS: time of lookup spent in physical file system
Contribution

• We argue that instead of using VFS caching for metadata, one can directly access metadata in physical file system
  • We took path lookup as a case to explore to remove VFS metadata caching for NVM FS
  • We implemented ByVFS, an optimization of VFS by removing dentry metadata caching in VFS
  • Result shows that ByVFS reduces application execution time by up to 26.9%
Outline

• Background & Motivation
• Design & Implementation
• Evaluation
• Related Work
• Summary
Virtual File System

- Caching metadata
  - Superblock
    - File system metadata
  - Dentry(cache in VFS named dcache)
    - File name and corresponding inode number
  - Inode (cache in VFS named icache)
    - File properties, such as file size, access time

- Supporting multiple file systems
- Concurrency control
- Permission check
Path Lookup Process in VFS

• Stat(“/home/wy/example.txt”)  
  1. Lookup dcache in VFS  
  2. Lookup dentry in physical file system  
  3. Lookup icache in VFS(hard link)  
  4. Lookup inode in physical file system  
  5. Create and initialize VFS structures  
  6. File access permission check  
  7. Judge the end of lookup  

• File lookup end condition  
  • Warm cache: step 1  
  • File does not exist: step 2  
  • VFS just contains icache: step 3  
  • Cold cache: step 5  

Lookup path is too long for cold cache
Bypass VFS(ByVFS)

• ByVFS
  1. Directly lookup dentry in physical file system
  2. Lookup inode in physical file system
  3. Get icache directly by pointer
  4. Create and initialize VFS structures
  5. File access permission check
  6. Judge the end of lookup

• File lookup end condition
  • File dose not exist: step 1
  • Warm cache: step 3 (lookup 2 times)
  • Cold cache: step 4 (lookup 3 times)

Remove the process of lookup dcache and icache
Remove dcache

• Issue 1
  • File name and pointer of icache
  • Solution
    • Get from physical file system

• Issue 2
  • Pointer of parent directory dcache (getcwd)
  • Solution
    • VFS
      • Iterate dcache of all path components to obtain the full path

1. dcache of “linux” gets dcache of “wy”
2. Repeat this process to get all dcache
3. Combine all file names of dcache

/home/wy/linux/ getcwd

We are here
Remove dcache

- Issue 1
  - File name and pointer of icache
  - Solution
    - Get from physical file system
- Issue 2
  - Pointer of parent directory dcache (getcwd)
  - Solution
    - VFS
      - Iterate dcache of all path components to obtain the full path
    - ByVFS
      - Use physical file system hardlinks "." and "..

We are here

```plaintext
/home/wy/linux/
```

```plaintext
/home/wy/linux/ getcwd
```

1 “.” get file “linux” inode number #3

```plaintext
/home/wy/linux/
```

2 “..” get parent directory (“wy”) data

```
. .. “linux” “xxx”
```

3 search inode name which inode number match #3

```
“linux”
```

```
/home/wy/
```

Repeat this process until to root directory
Keep icache

• icache
  • Frequently updated
  • Directly persisted to physical file system degrades the performance of the whole system

• Issue
  • How to locate icache
    • Add a pointer in inode to point to icache directly
    • Invalid when system crashes
    • Version number guarantees correctness
      • Superblock version number be updated when file system remounts
      • The pointer of inode is valid only when superblock version number equals inode version number
Discussion

• How to support multiple file systems in ByVFS
  • Add flag to the icache of the mounted directory
    • Once the flag `S_NVMFS_ROOT` is encountered, the remaining path lookup is performed in ByVFS

• Why not user-level file system
  • Without guaranteeing integrity, concurrency, crash-consistency and security
  • Developers are unwilling to modify POSIX-interface

• How to guarantee concurrency
  • Implement concurrency control in physical file system(future work)
Outline

• Background & Motivation
• Design & Implementation
• Evaluation
• Related Work
• Summary
Evaluation Methodology

• Platform
  • Server: Intel Xeon E5-2620 V3, 2CPU, each CPU has 12 cores
  • Emulating NVM using DRAM by adding write latency(600ns)
  • NVM 80GB
  • DRAM 16GB

• Compared System
  • Conventional VFS + physical file system NOVA[1](N)
    • Cold cache: VFS does not contain file metadata before evaluation
    • Warm cache: VFS contains file metadata before evaluation
  • ByVFS + modified NOVA(B)

[1] Xu, FAST 2016
Evaluation Methodology

• Benchmarks
  • System call
    • open, access, rename, unlink, mkdir, rmdir
  • Command-line applications performed on Linux source directory
    • du, find, ls, cp

• Metrics
  • VFS_lookup: the time spent on searching dcache or icache in VFS during path lookup
  • VFS_others: the time spent on other operations during path lookup
  • FS: the time spent in physical file system during path lookup
System Call—Cold Cache

Execution time are reduced by 7.4%~50.8%
System Call—Warm Cache

**Unlink**
- Require to operate in physical file system even in warm cache
- Execution time is reduced by 38.1%

**Access**
- VFS only needs to operate in VFS
- ByVFS needs to operates in physical file system
- Execution time is increased a little (4.7%)

ByVFS performs better for write operations, and achieve similar performance for read operations
Command-line Applications

- du/find/rm: execution time are reduced by 11.6%~26.9%
- ls/cp: execution time are increased by 3.8%~6.4%
  - Related with VFS warm cache hit ratios
  - We will optimize physical file system in the future
Outline

• Background & Motivation
• Design & Implementation
• Evaluation
• Related Work
• Summary
# Related Work

<table>
<thead>
<tr>
<th></th>
<th>Problems</th>
<th>Solution</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVM FS</td>
<td>Crash consistency</td>
<td>• Journal</td>
<td>PMFS[^2], NOVA[^4], NOVA-fortis[^5]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Copy-on-Write</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Log-structure</td>
<td></td>
</tr>
<tr>
<td>Software overhead</td>
<td>Bypassing page cache,</td>
<td>Bypassing page cache, block layer</td>
<td>BPFS[^1], PMFS[^2], SCMFS[^3], NOVA[^4], NOVA-fortis[^5], Aerie[^7]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimizing index structure</td>
<td>SCMFS[^3]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reducing kernel interaction</td>
<td>Aerie[^7]</td>
</tr>
<tr>
<td>Hardware overhead</td>
<td>Adding write buffer</td>
<td></td>
<td>HinFS[^6]</td>
</tr>
<tr>
<td>Metadata</td>
<td>Path lookup</td>
<td>Adding cache on top of VFS</td>
<td>[8,9]</td>
</tr>
<tr>
<td>operations</td>
<td></td>
<td>Full path lookup</td>
<td>[10,11,12,13]</td>
</tr>
<tr>
<td>Write amplification</td>
<td>Index optimization</td>
<td></td>
<td>[11,12,13,14,15]</td>
</tr>
</tbody>
</table>

\[^1\] Condit, SOSP 2009  
\[^2\] S.R. Dulloor, EuroSyS 2014  
\[^3\] Xu, FAST 2016  
\[^4\] Haris, EuroSys 2014  
\[^5\] [10] Lensing, SYSTOR 2013  
\[^6\] Zhan, FAST 2018  
\[^8\] Lu, FAST 2014  
\[^9\] [12] Yuan, FAST 2016  
\[^10\] Ren, ATC 2013
Outline

• Background & Motivation
• Design & Implementation
• Evaluation
• Related Work
• Summary
Summary

- Existing NVM based file systems keep VFS. However, VFS results in high metadata operation overhead
- ByVFS is an optimization of VFS by removing dentry caching to reduce extra overhead for NVM based file system
- Future work
  - Concurrency control
  - Optimizing metadata operations in physical file system
    - Highly efficient index structure
Thanks for your listening!

Q & A