DenseFS: A Cache-Compact Filesystem

**Zev Weiss, Andrea C. Arpaci-Dusseau,
Remzi H. Arpaci-Dusseau**

July 9, 2018

Filesystems in light of NVM

Existing filesystems mostly disk/flash-oriented

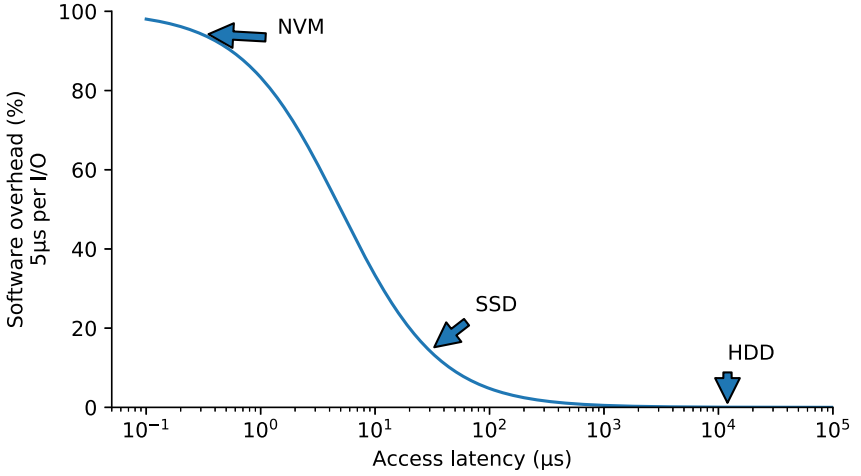
- Large storage unit
- Access latency ≫ memory

Nonvolatile memory (NVM) beginning to arrive

- Small storage unit
- Access latency ~= memory

Problem: many assumptions of FS design inverted

Performance inversion



Software CPU performance becomes critical

Goals:

- Understand sources of overhead in existing software
- Implement lightweight, low-overhead filesystem

Software performance

Major factor: **CPU cache**

- Fixed hardware resource (32KiB)
- Shared by all code that runs on CPU
  - Different pieces compete, interfere with each other

↳ Executing kernel code **degrades** application performance

Outline

- **Background**
- **Analysis**
- **DenseFS**
- **Evaluation**
- **Conclusions**

Analysis

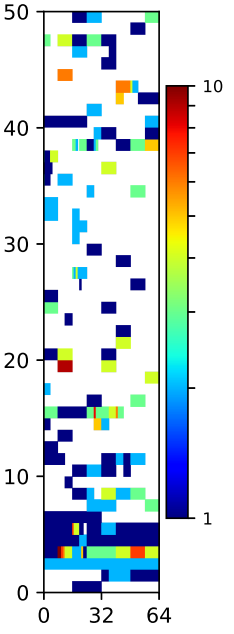
First step: examine cache behavior of existing filesystems

- btrfs
- ext4
- f2fs
- xfs
- tmpfs

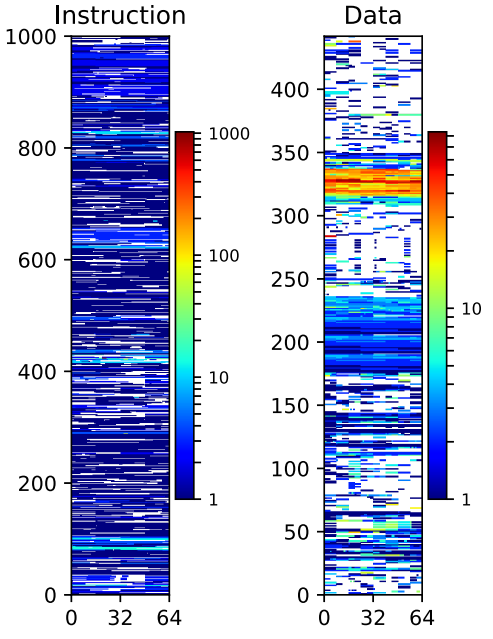
Methodology: **dynamic instruction tracing**

- Every kernel instruction executed
- Size & address of every memory access (code & data)
- Full stack backtraces
- Visualizations: cachemaps, cgstacks

Visualization: cachemaps

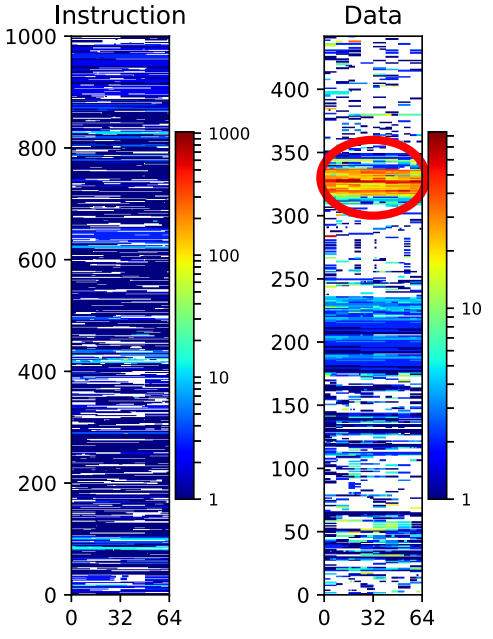


Visualization: cachemaps



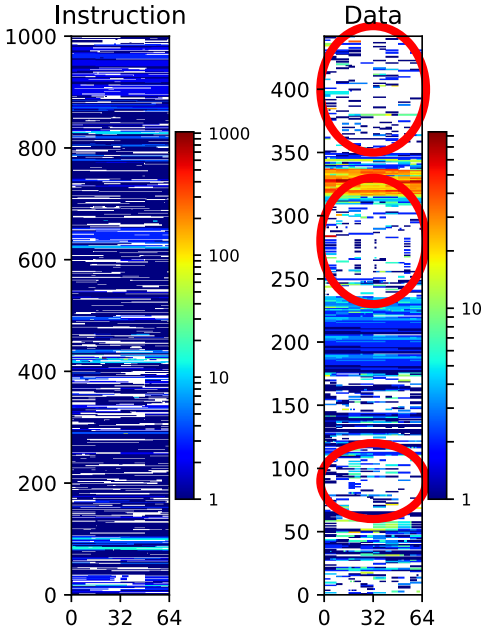
`creat` on xfs

Visualization: cachemaps



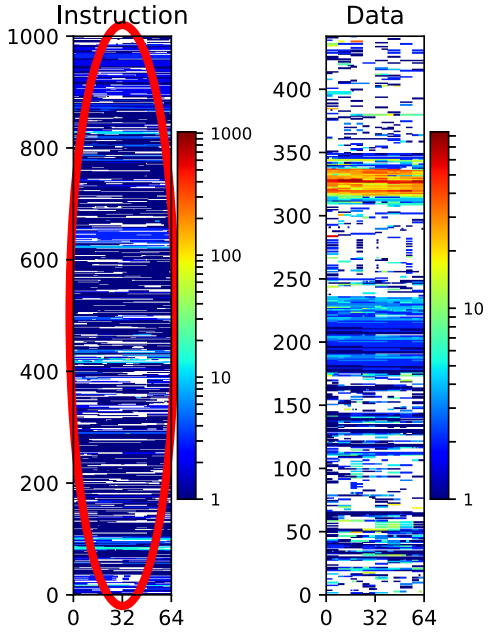
`creat` on xfs

Visualization: cachemaps



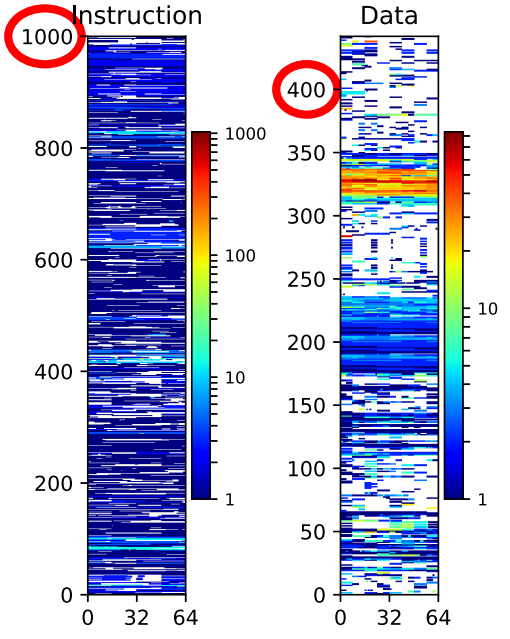
`creat` on xfs

Visualization: cachemaps



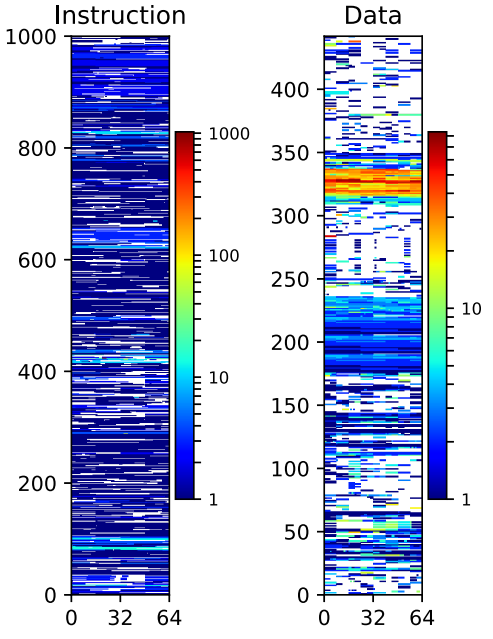
`creat` on xfs

Visualization: cachemaps



`creat` on xfs

Visualization: cachemaps



`creat` on xfs

Visualization: cgstacks

Coarse-**g**rained **stack** traces

Classify source locations into categories:

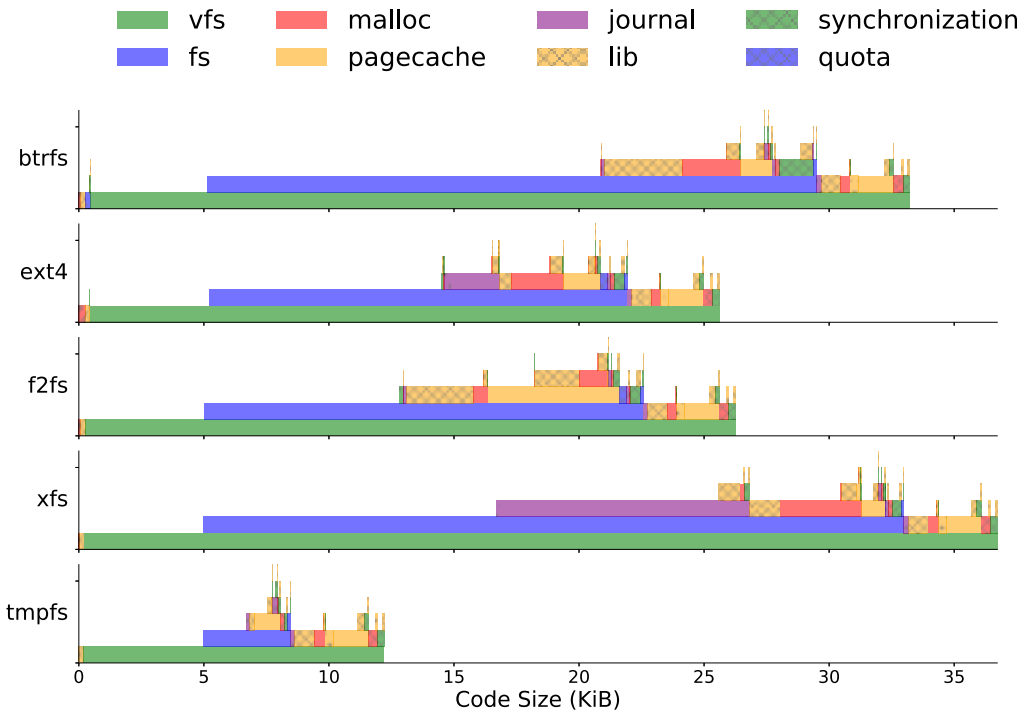
- VFS
- memory allocation
- pagecache
- ...

Coalesce backtraces by category

Result: high-level "origin" of each instruction

- e.g., "pagecache code called by VFS code"

Visualization: cgstacks



`creat`

Observations

Data:

- Large footprint (> ½ L1)
- Little spatial locality

Code:

- Even larger footprint (> L1)
- Little reuse
- Sizable overhead for generic FS infrastructure

DenseFS

Experimental Linux filesystem aiming for compactness

In-memory (pseudo-NVM)

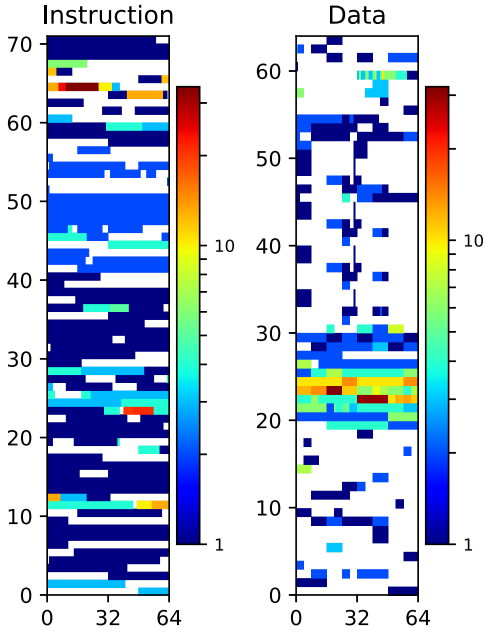
Simple structure

- Short code paths
- ~2.5KLoC

Avoids VFS & pagecache entirely

- Dedicated syscalls (`dfs_open()`, `dfs_read()`, etc.)
- Dedicated file descriptor table per process

DenseFS cachemaps



`creat`

DenseFS: inode compaction

Initial inode struct: essentially `struct stat`

```
struct dfs_inode {
    uint16_t nlink;
    ino_t inum;
    uid_t uid;
    gid_t gid;
    mode_t mode;
    off_t size;
    struct timespec mtime, ctime, atime;
    union {
        struct list_head dirents;
        struct rb_root chunks;
    } data;
    refcount_t pincount;
    spinlock_t lock;
};
```

112 bytes → **two** cache lines

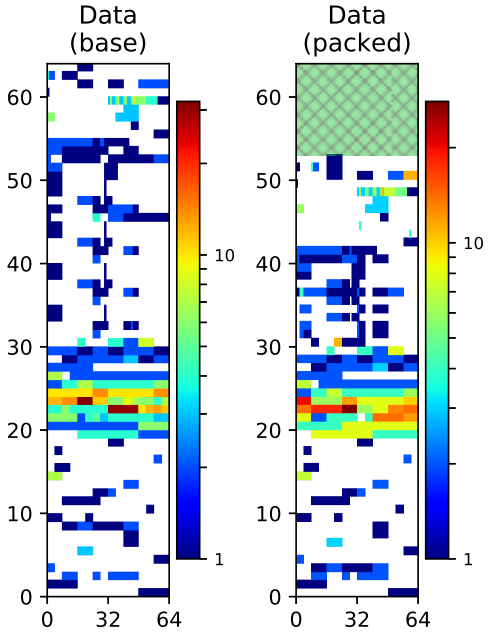
DenseFS: inode compaction

Modifications:

```
+struct imeta { uid_t uid; gid_t gid; mode_t mode; };

+/* FS-wide */
+struct { struct imeta* arr; uint16_t num; } imeta;

 struct dfs_inode {
     /* ... */
-    ino_t inum;

-    uid_t uid;
-    gid_t gid;
-    mode_t mode;
+    uint16_t meta_idx; /* index into imeta.arr */

-    struct timespec mtime, ctime, atime;
+    ktime_t mtime, ctime;
     /* ... */
 };
```

Size reduced to 56 bytes → **one** cache line

DenseFS: data compaction results

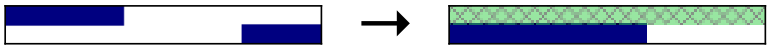


`creat`

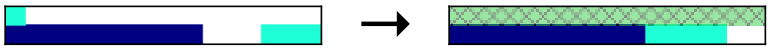
DenseFS: code compaction

Three techniques:

- Function alignment:

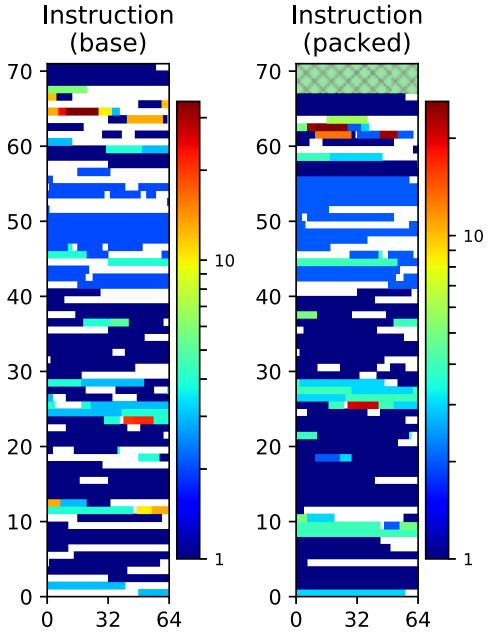


- Branch hinting:



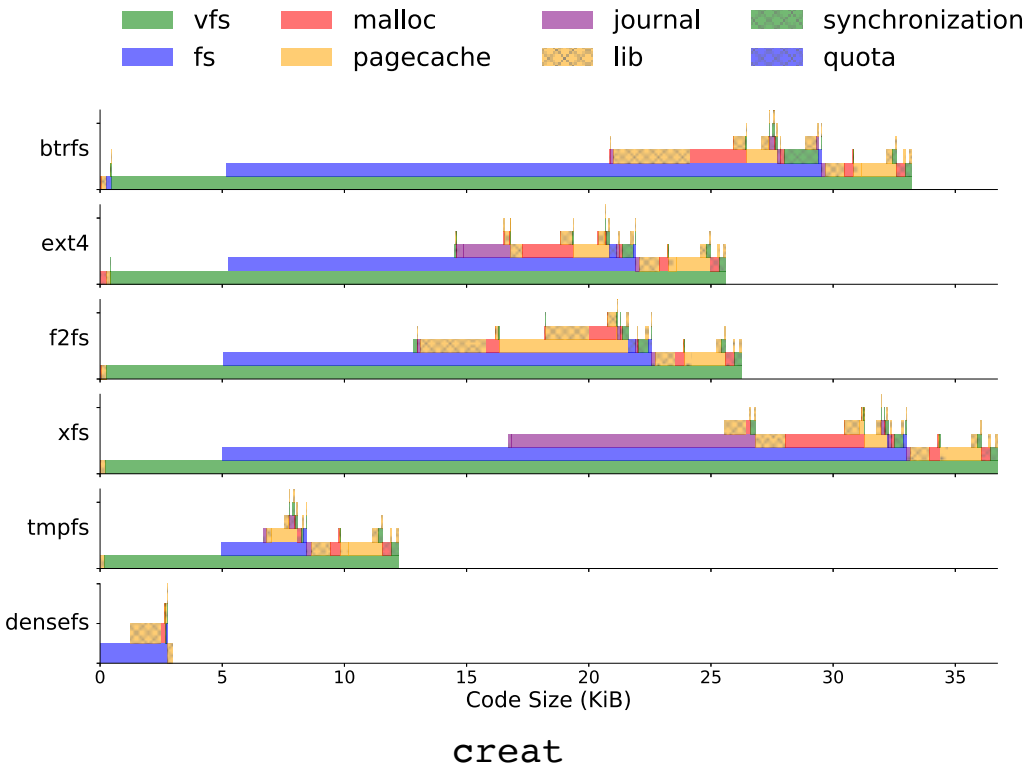
- Function ordering:

DenseFS: code compaction results



`creat`

DenseFS: cgstacks



DenseFS is smaller than generic VFS code alone

DenseFS microbenchmark

Specialized DenseFS-aware microbenchmark

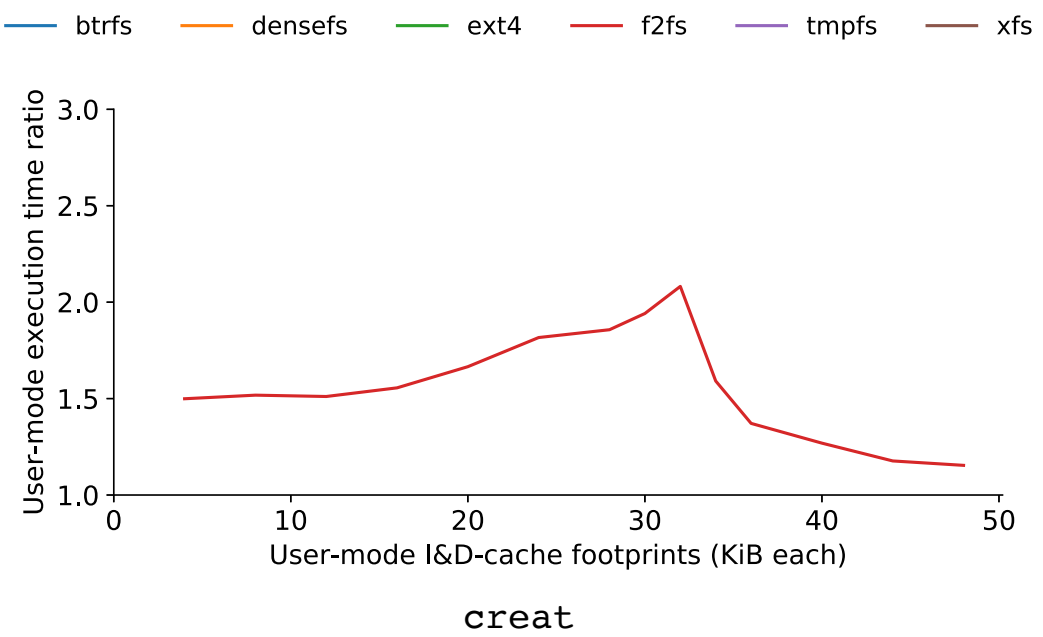
Inner loop:

- variable-footprint user code
- optional syscall

Measures performance of user and kernel execution independently

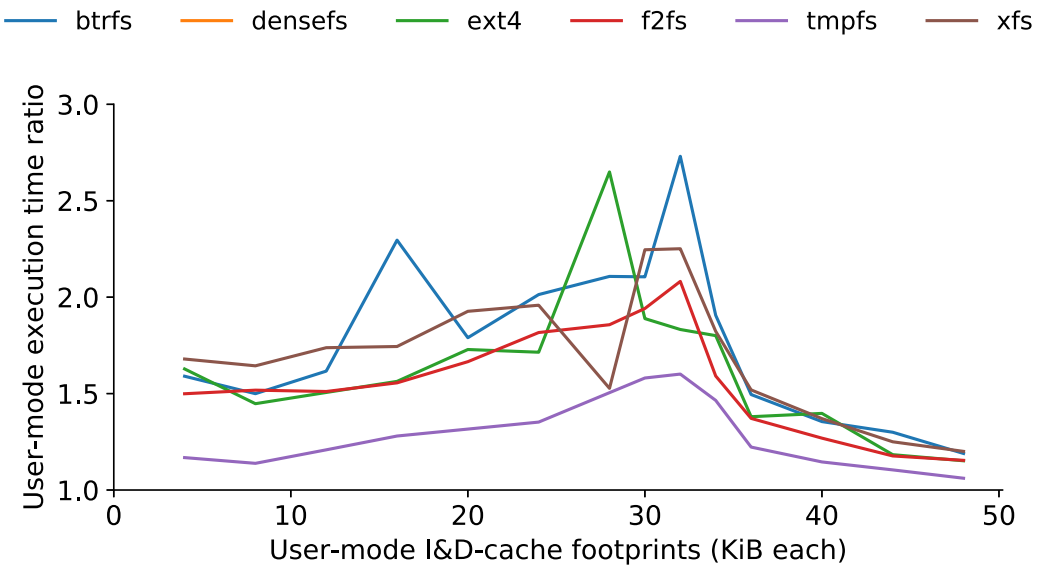
DenseFS microbenchmark

Impact of syscalls on user-mode performance:



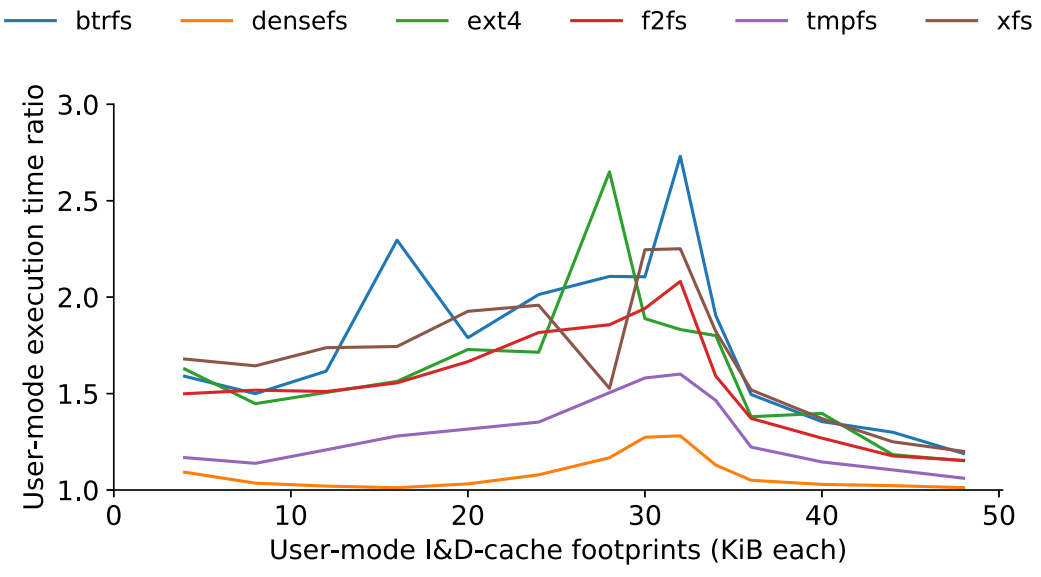
DenseFS microbenchmark

Impact of syscalls on user-mode performance:



DenseFS microbenchmark

Impact of syscalls on user-mode performance:



`creat`

Reduced cache pollution,
increased user-mode performance

DenseFS Conclusions

NVM inverts performance bottlenecks

- Software overhead dominates

Existing filesystems are large and heavy

- Slows execution of user code

DenseFS demonstrates performance potential of a much smaller, lighter filesystem:

- 170% performance penalty reduced to 30%
- 13-18% user-mode IPC increase

fin