

# Asynchronous Directory Operations in CephFS

Jeff Layton <[jl原因ton@redhat.com](mailto:jl原因ton@redhat.com)>

Patrick Donnelly <[pdonnell@redhat.com](mailto:pdonnell@redhat.com)>

# WHO ARE THESE GUYS?

- Jeff
  - longtime kernel dev for RH, focusing on network filesystems (NFS and CIFS, mostly)
  - has done some recent work with userland ceph
  - recently took over upstream maintainership of kcephfs
- Patrick
  - Joined RH in 2016; CephFS team lead
  - Works on all aspects of CephFS but mostly shepherds projects now.

# NETFS DIRECTORY OPERATIONS ARE SLOW

- `open(..., O_CREAT)`, `unlink()`, etc.
- usually involve a synchronous round trip to server
- Affects many common workloads:
  - untar'ing files
  - rsync
  - removing directories recursively
  - compiling software

# Observation: why are local file systems so fast?

- Obvious: no roundtrip latency with a remote file server.
- Local file systems **buffer metadata mutations in memory** until fsync on the directory/file or sufficient time has elapsed. Consequences:
  - Mutations can be written in batch to the journal in more efficient writes.
  - Operations are not guaranteed to be durable if no fsync is called:
    - rename, unlink, create require fsync on the containing directory file descriptor!
    - chmod, chown, setxattr require fsync on inode's file descriptor!

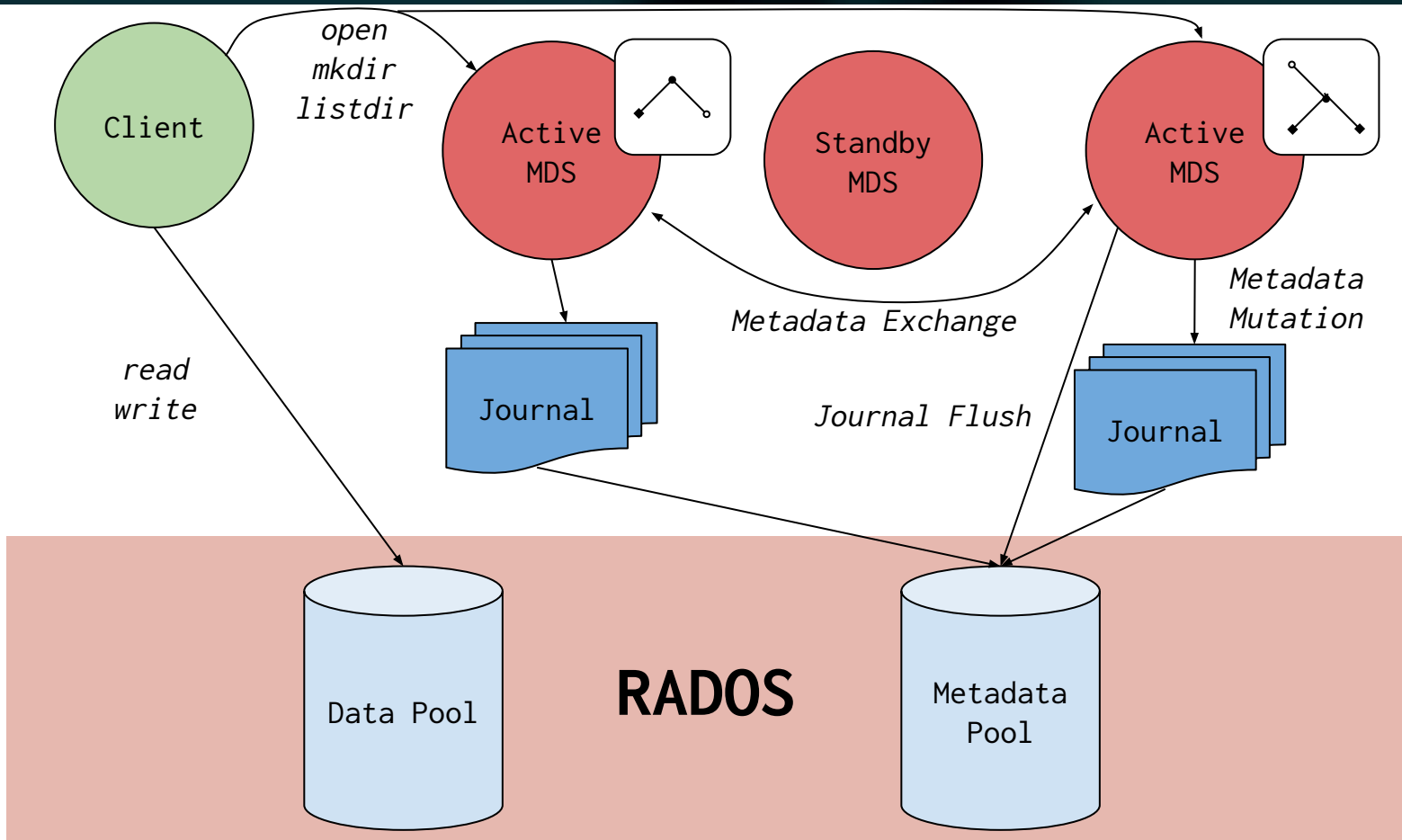
# Observation: why are local file systems so fast?

- Obvious: no roundtrip latency with a remote file server.
- Some (non-journaled) local file systems **buffer metadata mutations in memory** until fsync on the directory/file or sufficient time has elapsed.

## Consequences:

- Mutations can be written in batch to the journal in more efficient writes.
- Operations are not guaranteed to be durable if no fsync is called:
  - rename, unlink, create require fsync on the containing directory file descriptor!
  - chmod, chown, setxattr require fsync on inode's file descriptor!

# What's CephFS?



# CEPHFS CAPABILITIES

- CephFS capabilities (aka caps) delegate parts of inode metadata to client
- Types: PIN, AUTH, FILE, LINK, XATTR
- All have a SHARED/EXCLUSIVE variety
- FILE caps have other bits (READ, WRITE, CACHE, BUFFER, LAZYIO)
- Shorthand notation: **pAsxLsxFsxrcbIXsx**

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| p | _ | A s   x | L s   x | X s   x | F s   x   c   r   w   b   a   l |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```



# EXTENDING DIRECTORY CAPABILITIES

- FILE caps are largely unused on directories, except (Ds)
- Start handing them out on directories, and just interpret them differently
- So far:
  - CREATE requires Dc (aka Fc)
  - UNLINK requires Du (aka Fr)
- Work in conjunction with Fx caps
- Internally in MDS, done via a new lock caching facility
- Only handed out in response to first create or unlink in a directory
  - First call must be synchronous to establish the lock cache

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| p | _ |As  x |Ls  x |Xs  x |Ds  x  c  u                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

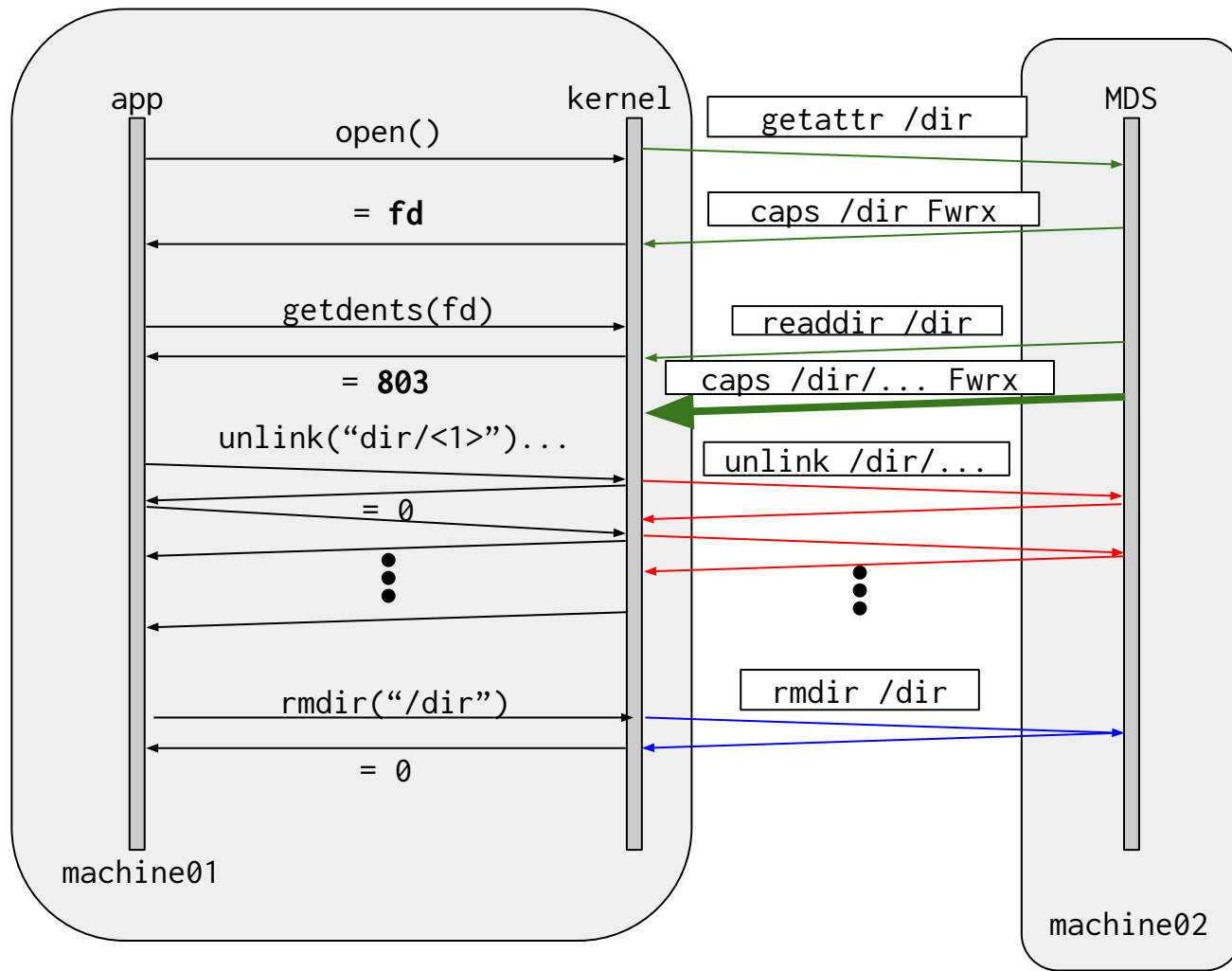
# CEPHFS DIRENT CACHING

- Async dirops require reliable cached information about dentry
- Two mechanisms
  - individual positive or negative dentry lease
  - Fs caps on directory
- For latter, also track directory “completeness”
  - Basically whether we have a record of all dentries in a directory
  - Allows us to satisfy negative lookups w/o talking to the MDS

# Asynchronous Metadata Mutations

# SYNCHRONOUS UNLINKS (STATUS QUO)

- In CephFS, unlink is done synchronously by the client. The application does not return from the syscall until the unlink is durable.
- This is particularly slow for recursive unlinks...

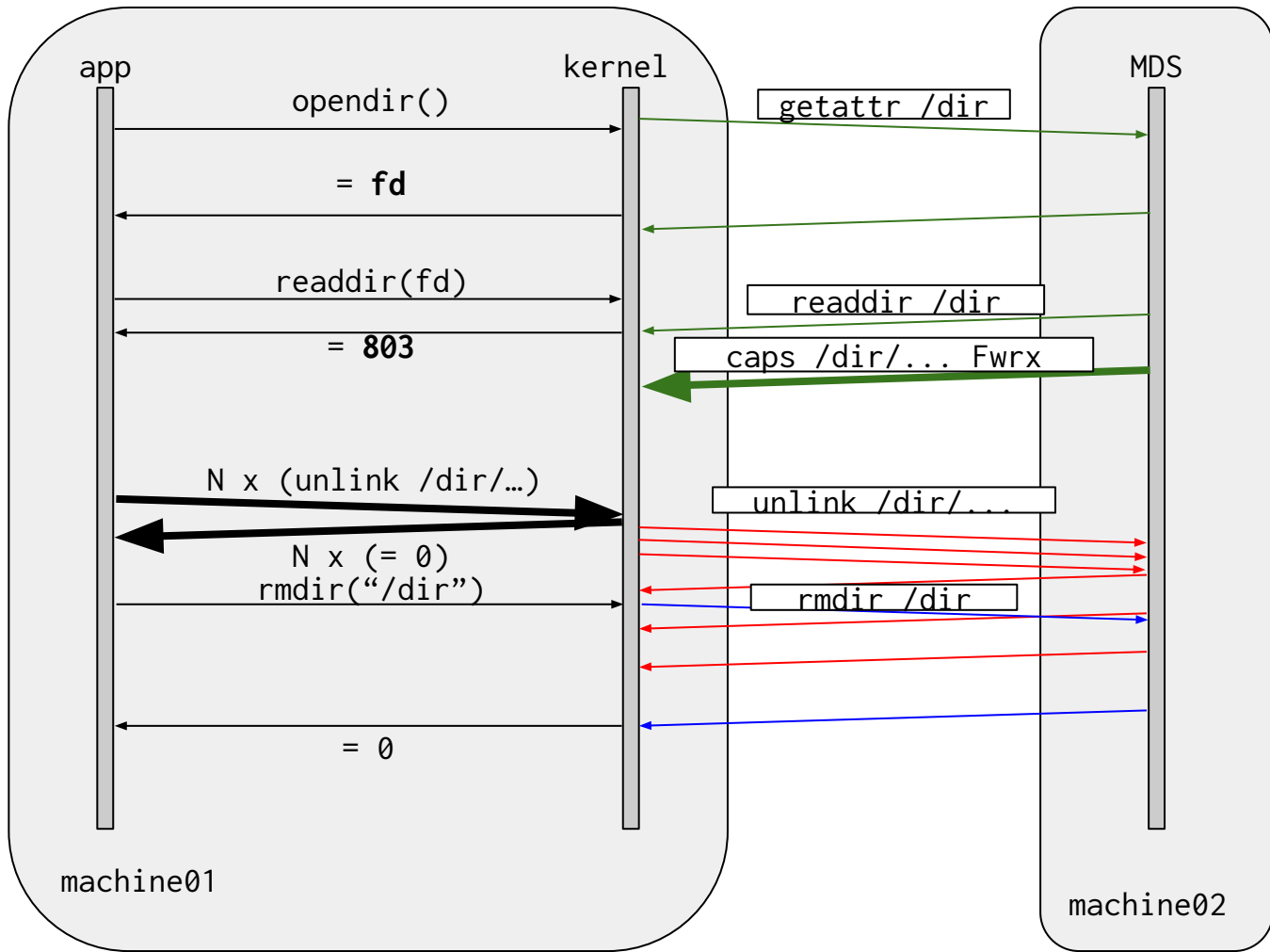


# SHOULD WE WAIT TO TRANSMIT ASYNC REQ'S?

- Namespace ops are fundamentally different from data writeback
- Normal Files:
  - Data is stored in kernel's pagecache and later written to backing store
  - High probability that recently modified data will be changed again in future
  - Advantageous to delay write requests for a bit to allow writes to be batched
- Directories:
  - Workloads that rapidly create/unlink same dentry are fairly rare
  - Not much advantage to delaying transmission of any async request (exception: rsync)
  - Might change if we deem batched calls to be useful in future

# ASYNCHRONOUS UNLINK

- Requirements:
  - Fx and Du (aka Fr) caps on parent directory
  - Known positive dentry
    - Positive dentry lease
    - Fs on the directory and a positive dentry
- Fire off UNLINK call to MDS and then immediately delete the dentry locally
- When reply comes in, do only minimal processing
- rmdir() of parent has to wait for all child unlink replies to come back
  - Future work: support async rmdir!





# UNLINK PERFORMANCE

Where the test-dirops directory has 10k files:

```
$ time rm -f /mnt/cephfs/test-dirops/*
```

Without async dirops:

```
real    0m10.371s
user    0m0.138s
sys     0m0.672s
```

With async dirops:

```
real    0m0.385s
user    0m0.110s
sys     0m0.077s
```

# TIME SPENT IN ceph\_unlink()

Without async dirops:

```
@unlink[rm]:
[512K, 1M)          7855
[1M, 2M)           2033
[2M, 4M)            92
[4M, 8M)             5
[8M, 16M)           5
[16M, 32M)          2
[32M, 64M)          8
```

With async dirops:

```
@unlink[rm]:
[1K, 2K)            3380
[2K, 4K)            3348
[4K, 8K)            3182
[8K, 16K)           65
[16K, 32K)          19
[32K, 64K)          0
[64K, 128K)         0
[128K, 256K)        1
[256K, 512K)        1
[512K, 1M)          1
[1M, 2M)             1
[2M, 4M)             1
[4M, 8M)             0
[8M, 16M)            1
```

# OPPORTUNITIES TO IMPROVE UNLINK

- Asynchronous rmdir
  - rmdir acts as an implicit fsync, preventing continuation until all child dirents are unlinked
  - `rm -rf /mnt/cephfs/test-dirops/` behaves differently!
- Tuning in-flight asynchronous unlink operations
  - Find the proper balance between slowing down the application and performing the unlinks as fast as possible. Too many operations in flight may disrupt other applications or other CephFS clients!
- Batching unlink operations
  - Gather up unlink operations into single RPC so MDS can more efficiently acquire locks and write journal segments.

# ASYNCHRONOUS CREATE

- Requirements:
  - Fx and Dc (aka Fc) caps on parent directory
  - Known negative dentry
    - Negative dentry lease
    - Fs on parent directory + completeness
  - File layout (copied from first sync create in a directory)
  - Delegated inode number
- Fire off the create call immediately set up new inode and return from open()
- Assume newly-created inode gets full caps from MDS (pAsxLsxFsxcbrwXxs)
- Always set O\_EXCL in the call to MDS

# INODE NUMBER DELEGATION

- Need to know in advance what the inode number will be
  - to hash inode properly in kernel
  - allow for writes before reply comes back: solves fsync races
- MDS will now hand out ranges of inode numbers in CREATE responses
- new userland tunable: `mds_client_delegate_inos_pct`
  - “percentage of preallocated inos to delegate to client”
  - default == 50, so client usually has ~500 at a time
- Tied to MDS session
  - if session is reconnected, then (WIP) client should resend async creates with previously delegated inodes

# CREATE PERFORMANCE

Create 10k files in a directory:

```
time for i in `seq 1 10000`; do
    echo "foobarbaz" > $TESTDIR/$i
done
```

Without async dirops:

```
real    0m11.390s
user    0m0.315s
sys     0m0.974s
```

With async dirops:

```
real    0m5.519s
user    0m0.132s
sys     0m0.496s
```

# TIME SPENT IN ceph\_atomic\_open()

Without async dirops:

```
@open[test-async-diro]:
[256K, 512K)      8
[512K, 1M)       9791
[1M, 2M)         187
[2M, 4M)          2
[4M, 8M)          0
[8M, 16M)         0
[16M, 32M)        0
[32M, 64M)        1
[64M, 128M)       9
[128M, 256M)     2
```

With async dirops:

```
[8K, 16K)        641
[16K, 32K)       2388
[32K, 64K)       6290
[64K, 128K)      599
[128K, 256K)     52
[256K, 512K)     1
[512K, 1M)       0
[1M, 2M)         6
[2M, 4M)         3
[4M, 8M)         0
[8M, 16M)        0
[16M, 32M)       0
[32M, 64M)       0
[64M, 128M)      0
[128M, 256M)    14
[256M, 512M)     6
```

??



# Kernel Build (time make -j16 ; time make clean)

```
#!/bin/bash

mkdir linux
cd linux/
tar xf ../linux.tar
make defconfig
make -j16
```

Without async dirops:

```
real    4m57.678s
user    26m43.167s
sys     4m21.124s
```

With async dirops:

```
real    4m6.937s
user    25m47.064s
sys     3m58.909s
```



# OPPORTUNITIES TO IMPROVE CREATE

- Optimize for rsync
  - In-place renames
- Batching creates similar to unlink
- Other operations: mkdir, symlink, in-place rename
- Error handling...

# ERROR HANDLING

- If we return early from `unlink()` or `open()`, then what to do when the ops fail?
  - For creates, we may have already closed the file by the time reply comes in
  - Which failures are permitted by the protocol?

- From `fsync(2)` manpage:

Calling `fsync()` does not necessarily ensure that the entry in the directory containing the file has also reached disk. For that, an explicit `fsync()` on a file descriptor for the directory is also needed.

- Nobody really does this, and most modern local fs' journal the create

# ERROR HANDLING (CONT'D)

- Currently after failed unlink
  - mark directory non-complete
  - invalidate dentry
  - set writeback error on parent directory to show up on fsync(dirfd)
- After failed create
  - invalidate dentry
  - set writeback error on parent directory
  - set writeback error on created inode
- One idea: propagate errors to parent directories all the way up to the root of the mount
- We may need to consider new interfaces

# Questions?

Jeff Layton <[jlayton@redhat.com](mailto:jlayton@redhat.com)>

Patrick Donnelly <[pdonnell@redhat.com](mailto:pdonnell@redhat.com)>

<https://ceph.io/>

<https://github.com/ceph/ceph.git/>