DIFFERENCES FROM POSIX

CephFS aims to adhere to POSIX semantics wherever possible. For example, in contrast to many other common network file systems like NFS, CephFS maintains strong cache coherency across clients. The goal is for processes communicating via the file system to behave the same when they are on different hosts as when they are on the same host.

However, there are a few places where CephFS diverges from strict POSIX semantics for various reasons:

- If a client is writing to a file and fails, its writes are not necessarily atomic. That is, the client may call write(2) on a file opened with O_SYNC with an 8 MB buffer and then crash and the write may be only partially applied. (Almost all file systems, even local file systems, have this behavior.)
- In shared simultaneous writer situations, a write that crosses object boundaries is not necessarily atomic. This means that you could have writer A write “aa|aa” and writer B write “bb|bb” simultaneously (where | is the object boundary), and end up with “aa|bb” rather than the proper “aa|aa” or “bb|bb”.
- Sparse files propagate incorrectly to the stat(2) st_blocks field. Because CephFS does not explicitly track which parts of a file are allocated/written, the st_blocks field is always populated by the file size divided by the block size. This will cause tools like du(1) to overestimate consumed space. (The recursive size field, maintained by CephFS, also includes file “holes” in its count.)
- When a file is mapped into memory via mmap(2) on multiple hosts, writes are not coherently propagated to other clients’ caches. That is, if a page is cached on host A, and then updated on host B, host A’s page is not coherently invalidated. (Shared writable mmap appears to be quite rare–we have yet to here any complaints about this behavior, and implementing cache coherency properly is complex.)
- CephFS clients present a hidden `.snap` directory that is used to access, create, delete, and rename snapshots. Although the virtual directory is excluded from readdir(2), any process that tries to create a file or directory with the same name will get an error code. The name of this hidden directory can be changed at mount time with `-o snapdirname=.somethingelse` (Linux) or the config option `client_snapdir` (libcephfs, ceph-fuse).

PERSPECTIVE

People talk a lot about “POSIX compliance,” but in reality most file system implementations do not strictly adhere to the spec, including local Linux file systems like ext4 and XFS. For example, for performance reasons, the atomicity requirements for reads are relaxed: processing reading from a file that is also being written may see torn results.

Similarly, NFS has extremely weak consistency semantics when multiple clients are interacting with the same files or directories, opting instead for “close-to-open”. In the world of network attached storage, where most environments use NFS, whether or not the server’s file system is “fully POSIX” may not be relevant, and whether client applications notice depends on whether data is being shared between clients or not. NFS may also “tear” the results of concurrent writers as client data may not even be flushed to the server until the file is closed (and more generally writes will be significantly more time-shifted than CephFS, leading to less predictable results).

However, all of there are very close to POSIX, and most of the time applications don’t notice too much. Many other storage systems (e.g., HDFS) claim to be “POSIX-like” but diverge significantly from the standard by dropping support for things like in-place file modifications, truncate, or directory renames.

BOTTOM LINE

CephFS relaxes more than local Linux kernel file systems (e.g., writes spanning object boundaries may be torn). It relaxes strictly less than NFS when it comes to multiclient consistency, and generally less than NFS when it comes to write atomicity.

In other words, when it comes to POSIX,

```
HDFS < NFS < CephFS < {XFS, ext4}
```