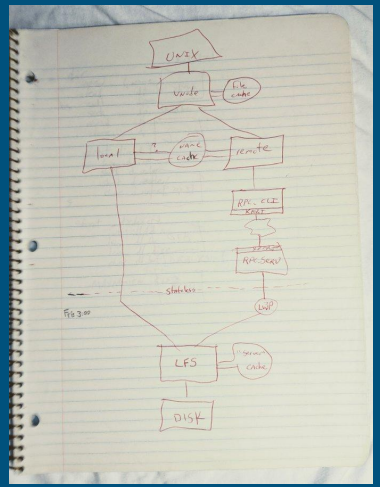
NFS Must Die!

And how to get beyond file sharing in the cloud.

Tom Lyon - @aka_pugs@mastodon.social

Who is Tom Lyon? (aka Pugs)

- Programming since 1967
- UNIX since 1975 - Princeton, Bell Labs, Amdahl, Sun…
- #8 at Sun Microsystems - NFS, Automounter, Translucent File System, NSE
- Founder/CTO/Chief Scientist of
  - Ipsilon Networks (-> Nokia 1998),
  - Netillion (->⏚ 2006),
  - Nuova Systems (-> Cisco 2009),
  - DriveScale (-> Twitter 2021)
- Mostly retired 2022



Datasets

This talk is about networks and datasets, especially in the cloud

A dataset is a set of related files with some application-determined consistency among the files

In Linux, can consider a dynamically mountable file-system as a dataset

This talk is not just about “The” NFS, but *any* NFS

Target/Example Use Case

AI training dataset with a billion files

A few hundred agents frequently updating the dataset with a few thousand files each time

Thousands of GPU systems consuming the dataset

NFS: The Good Parts

- Rapid access to arbitrarily large datasets
  - No need to copy first!
- Access by *network-oblivious* programs - they think it's local data
  - Network aware programs should use object storage
- Widely implemented
- Amazing engineering, getting better all the time, but…

Existential Crisis: Shared Mutable Data

Most important lesson in concurrent & distributed programming:

Shared Mutable Data is a BAD IDEA

What is the purpose of a network file system?

NFS provides Shared, Mutable Data

There is no ‘N’ in POSIX

- POSIX file operations assume consistency and availability
  - Consistency - concurrent clients see same data/results
  - Availability - if the file system is unavailable, your client is also dead
- CAP Theorem
  - In a network, you can’t have both Consistency and Availability
  - Partitions happen
- NFS hard mounts (Cons.) vs soft mounts (Avail.)

Semantics? What Semantics?

- POSIX? (Have *you* read the spec?)
- Protocol?
- Particular filesystem?
- Syscall interface?
- Every system chooses a subset of the above, and then adds special features
- Every interface constantly evolving (protocols slowly)
- Result: Mine field of unsafe practices

2 Hardest Problems in Computer Science

1. Naming
2. **Cache Invalidation**
3. Off-by-one errors

Caching is **necessary** for performance

Cache invalidation: slow AND buggy!

Who is the Client?

- Traditional: People
  - Care a lot about names
  - Not Restartable
- Cloud: Functions (jobs, containers, serverless functions, whatever)
  - Just want a handle/uuid
  - Restartable (pure functional) is goal
- Developer: every machine is like my machine, right?

This Is The Way: Layered, Immutable Data

- Cloud scale dataset success stories
  - Git
  - Docker files
  - Delta Lake, Pachyderm, LakeFS, many more
- Data CANNOT be shared unless/until it is immutable (commit point)
- Cache it! Replicate it! It won’t change!
- BUT…
  - Too much copying!!

NVMe Over Fabrics & Block Storage Providers

- Crazy fast remote block device access.
- Block semantics - very well defined!
- Blocks are aggressively cached by OS
- BeyondFS allows many different block storage providers
  - AWS EBS, Google & Azure equivalents
  - OpenEBS, Longhorn, Ceph
  - DellEMC, NetApp, Pure
  - LVM or ZFS volumes for easy dev/test
  - Even files emulating block devices

Device Mapper Magic

- Snapshot - copy on write
- Extend - online FS expansion - no need to hit ENOSPC
- Multi-pathing - for better network availability
- Dm-crypt for privacy, dm-verity for integrity
- Many, many, more dm capabilities

The solution: Beyond FS (byfs)

Layered, immutable, block based file-system images

Accessed via network block protocols - nvme-o-f, iscsi, …

Copy-on-write semantics for updating

But, no updates shared until a commit point for the dataset

Auto-mounted, cloud-scale naming & sharing

Per-domain control service to keep track of everything

Copy-on-Write: Files or Blocks?

- OverlayFS - CoW for files - good for most types of files
- DM Snapshot - CoW for blocks - better for databases, log files, etc.
- Ext2/3/4 FS in each layer - Linux “native”, *some* support in every OS
- 2 regions in each layer - 1 for files, 1 for DB (manual placement req’d)
- DB region also used for …

The Billion File Problem

- On very large FSes, users running *find* can bring serious pain
- Change *find* to look for an actual database at the top of the block snapshot region
- Update on each commit
- Make it serious with SQLite3 - real SQL queries

The Small File Problem

- Every file comes with a huge amount of metadata
- Many files + attempted consistency leads to terrible performance
- Journaling file-systems cause more delays
- In BeyondFS, *files* never cross the wire, only *blocks*
- Journaling not needed, sync/umount is only consistency point.
- Faster than “local” and Free transactions!

Security

- SSH-like key management for Authentication & Authorization
- Within a dataset, all files are readable
- Mount-time translation of userids to “nobody” equivalent
- Dm-crypt for at-rest and on-the-wire privacy
- Dm-verity for strong checksums

Naming

- For functions:
  - /byfs/uuid[@domain]/x/y/z
    - Enters a writable space with RO layers below
- For people (hard problem):
  - /byfs/project[@domain]/dataset/x/y/z
- /byfs automount everywhere, including containers
- Well known DNS server/service: byfs.example.com

Creation

Byfs create [-C class] [Human name]

- Returns uuid to empty dataset; uuid always means “latest”

Class determines

- Storage backend
- Replication/Retention policies
- Authorization

Lots of commands to maintain the human friendly name space

Commits & Conflicts

“Byfs commit” - commit current dataset, return uuid

- Merge conflicts - commit fails - restart function

Various merge policies possible

Ay, Ay, EIO

Even with block semantics, network partitions must be dealt with.

Network timeouts eventually cause I/O error - EIO

Only 1 program in existence that properly handles EIO

Kernel enhancement: mount flag MS_FAILHARD

- Any EIO on mount point causes all following I/Os to fail

Byfs refuses commit on any failed dataset

Garbage collection

Each layer keeps a reference count:

- Upper layer(s)
- Current users (lease based w timeouts for lost clients)
- Tags

Garbage collection -

- If no tags or active users, a layer may be merged into the next layer up

Lazy unmounts

Don’t unmount layers aggressively

- High potential for layer reuse
- Blocks remained cached

Summary

Network file systems are inherently unsafe

The CAP theorem *will* bite you

No one can agree on which semantics are safe; no tools exist to check or constrain you to a safe subset

Block storage has well defined semantics and an immense number of under-utilized capabilities

Let’s get beyond…

Status

This is a call for collaborators.

I need help. Lots of help.