Lustre: building a cluster file system for 1,000 node clusters

Phil Schwan

phil@clusterfs.com

http://www.clusterfs.com

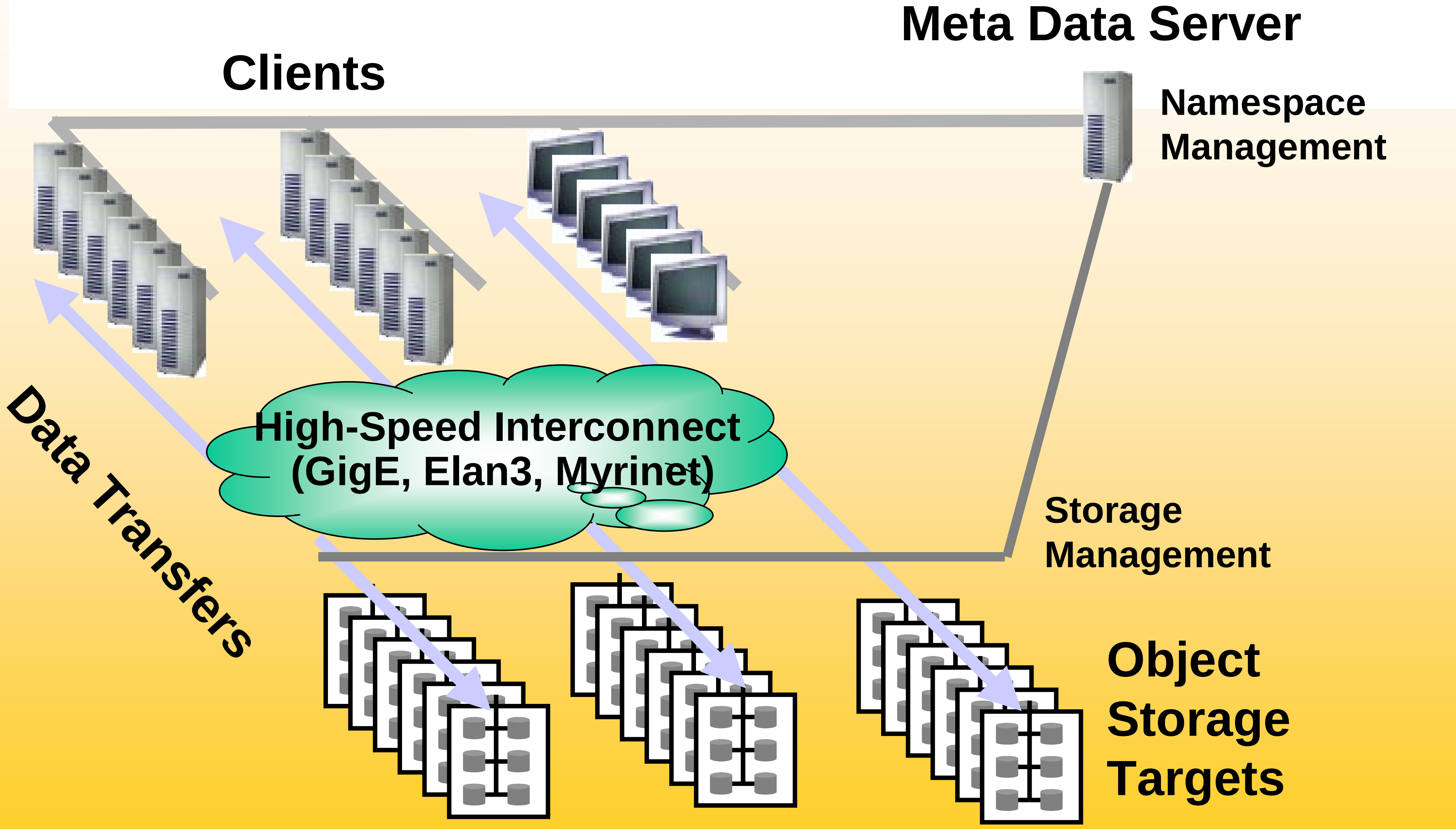
Cluster File Systems, Inc

Topics

- 60-second Lustre introduction
- A slice of what went well
- One critical mistake
- Questions
- Time permitting: on deck for 2003-2004

(very) Basics

- GPLed cluster file system for Linux
  - Stable on 2.4.x, making rapid progress for 2.6
- Aims for POSIX compliance
- Layering of object protocols
- Distributed lock manager
- Usually separate metadata and file data servers



Cluster File Systems, Inc

What went well

Distributed Lock Manager

Simplicity

- Based on VAX DLM concepts
- Built from scratch
- Why not use the IBM DLM?
  - 1/10th of the size (4,000 lines of code)
  - We don’t need most complicated features
  - We do need different extensions

DLM extensions

- Extent locks
  - A new lock type, with an extra field
  - Policy for automatic extent growth
  - Common case: one user, one lock
- Lots of file systems don’t manage this well
  - One lock per file – no concurrency
  - One lock per block/page – billions of locks
  - Locks? Where?

DLM extensions part 2

- Intent locking
  - Allows the DLM to make policy decisions
  - Grant a lock in a low-concurrency situation
  - Perform the operation in a high-concurrency situation
- More on intent locks coming up

What went well

Scaling metadata to 1,000 nodes

Scaling metadata to 1,000 nodes

- Consider: 2,000 processes on 1,000 nodes
- All create one or more files simultaneously
- In the same directory

- This is not contrived—some LLNL science runs do this every hour

Metadata option #1: lock the directory

- Take a write lock on the parent directory
- Check to see if the file exists
- Add the new directory entry

- Very efficient for the single-user case
- Easy to implement: mimics the VFS code
- A complete disaster for our 1,000-node example

Metadata option #2: raw calls

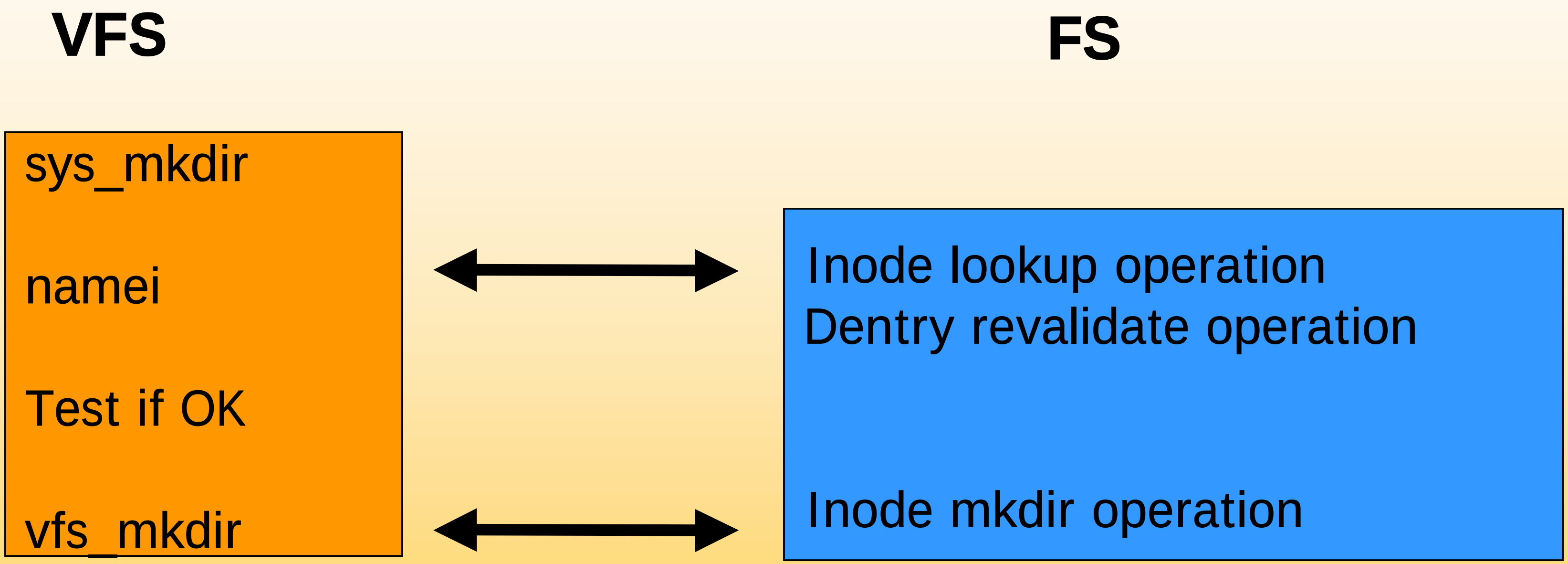
- Execute operations entirely on the server
- Don’t return locks to clients, only a status code

- Avoids lock ping-pong
- File creations can take one RPC
- Not very good for the single-user case

What went wrong

Metadata Intent Locking

Current Linux VFS



VFS

FS

sys_mkdir

namei

Test if OK

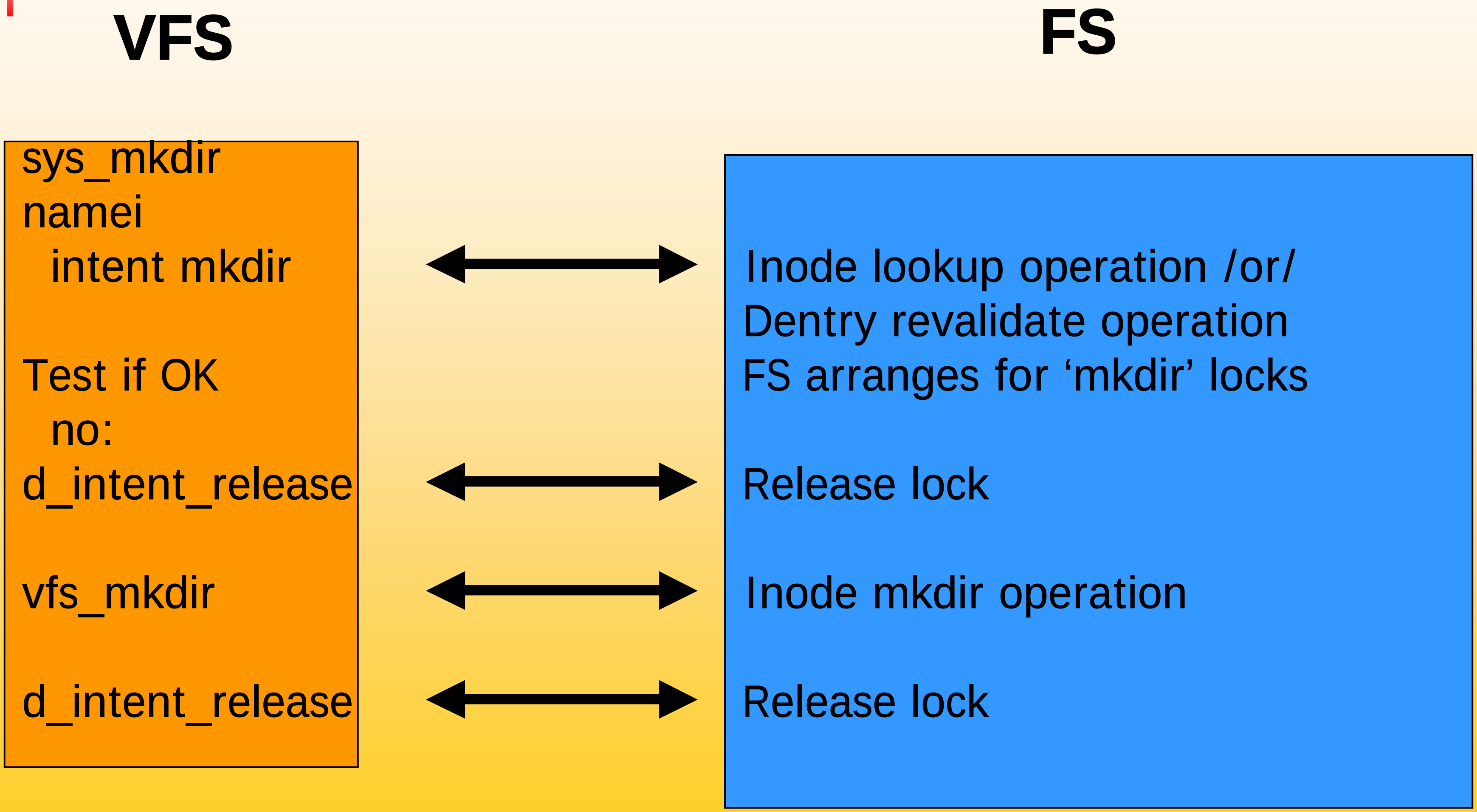
vfs_mkdir

Inode lookup operation

Dentry revalidate operation

Inode mkdir operation

We added “intents” to lookups



VFS

sys_mkdir
namei
  intent mkdir

Test if OK
  no:
d_intent_release

vfs_mkdir

d_intent_release

FS

Inode lookup operation /or/
Dentry revalidate operation
FS arranges for ‘mkdir’ locks

Release lock

Inode mkdir operation

Release lock

What’s the point?

- VFS code prefers to lock directories
  - The intent code reorganizes around that
- Not all metadata loads are alike
  - Locking directories is terrible for concurrent updates
  - Server execution is terrible for the common single-client case

- Intents give the option to the file system

Intents gone wrong

- Juggling too many things
  - Needed too many locks to safely use the VFS code
- Too many corner cases
  - Server view of execution must *exactly* match the client's view
- There was a simpler solution right around the corner…

Intents become “raw” operations

- Still gives the lock manager an opportunity to choose

- If contention is low…
  - Server returns a *write-back lock*
- If contention is high…
  - Server executes for us, sends a return code
  - Returns no locks at all
  - We skip **all** client VFS code

What went well

Object Protocols

Lustre file I/O in brief

- Object protocols were a no-brainer
- No shared-block file system will scale to 1,000 nodes
  - Shared disk much too expensive
  - Locking for block allocation
- Lustre storage targets manage object and block allocation

Lustre file I/O in brief

- Very simple object-based protocol. For example:
  - lock(object id, start, end) → returns lock handle
  - write(object id, offset, data, length)
  - unlock(lock handle)

Yes, but does it *work*?

What went so-so

Debugging

Debugging

- An extensive logging system
  - The log output is frequently more than the size of the I/O
  - Full debug is gigabytes for a simple test
- Tools to filter and contextualize the logs
- Using the logs requires immense understanding

“also, if you think the stuff in the Matrix about people learning to “read” is lame, then you haven’t watched Phil read Lustre debug logs.” – Jacob

Tools

- Linus hates them, but we thrive on good debug tools
- The first time we write a piece of code, we test it in UML under GDB
- We make extensive use of *mcore*, *netdump*, and *crash* on the real hardware
  - Saves more time than I care to count
- Working on kgdb-over-UDP extension
  - Turns out someone already started

Debug issues

- Sometimes we let our tools slip
  - Improving our tools almost immediately improves our work
  - The tools on ia64 were *terrible* for a long time
- It’s not a trivial system
- It’s still a 1,000-node state machine

- Overall, we get a B for debugging

The real world

- 3 of the top 8 supercomputers in the world run Linux. Lustre runs on all 3.
  - LLNL MCR: 1,100-node ia32 cluster (#3)
  - LLNL ALC: 950-node ia32 cluster (#6)
  - PNNL EMSL: 950-node ia64 cluster (#8)
- Installing in 2003-2004:
  - NCSA: 1,000 nodes
  - SNL/ASCI Red Storm: 8,000 nodes
  - LANL Pink: 1,000 nodes
- Chosen for ASCI PathForward SGS file system

Cluster File Systems, Inc