Failure Handling

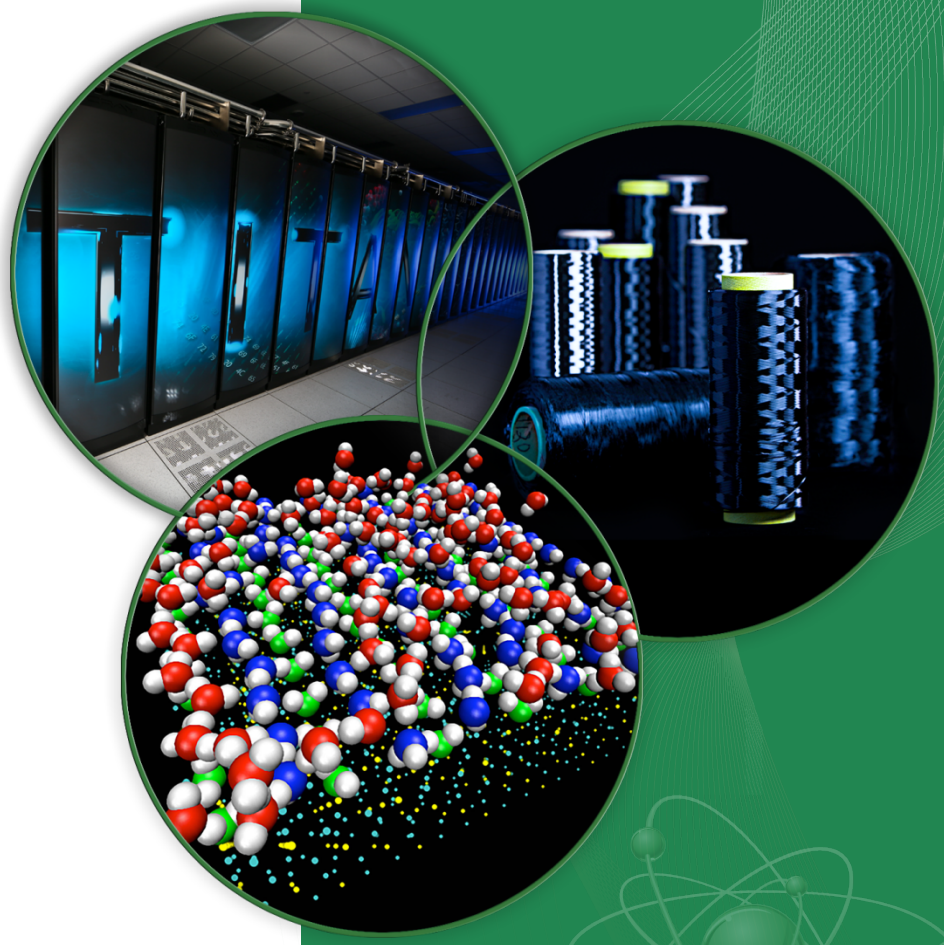
With Grace, hopefully…

Jason Hill

Storage Team Lead

OLCF, HPC Operations

3-3-2015



Overview

- Dirty laundry
- Design decisions
- 2am stupidity
- Specific design points
- A few case studies

Dirty Laundry

- We all have the horror stories
  - Corruption
  - Lost/destroyed files
  - Poor performance
- Why don't we learn from mistakes?
- How can we build a better community of administrators?

Learning from Mistakes

- One significant roadblock to sharing is the flexibility of Lustre
  - Hard to distill lesson learned to be applicable across all environments
  - Sysadmins are not always proactive
- Community is small
  - We've all seen some variant of X
  - This doesn't help create new talent
- Lots of "dark" community members
  - Secure sites can't share

Learning from Mistakes

- Develop lessons learned that are passed to management after incidents.
- Distilled from a detailed analysis of the event
  - Internal wiki document
- Don’t assign blame
  - It never helps
- Always build confidence in the team
  - This always helps

Design Decisions

- We make strategic design decisions
  - Limit the impact of individual failures
- These are always at odds with funding sources
  - Unlimited resources == no SPOF design ?!?!
- Here's where flexibility of Lustre makes things hard
  - Appliance
  - Vendor specific solutions
  - "roll your own"

Design Decisions

- No Single Point of Failures
- Limit component types
- Diskless provisioning
- Don’t duplicate services

2am Stupidity – a Design Principle

- It's not just for college students!
- Our design considerations take this into account
- KISS – not the rock band!
- Limited hardware variation helps here
- You're never a rocket scientist when you get woken up

Specific Design Points

- Worth reiterating: No SPOF's
  - Look at your storage subsystem closely
  - You have the least control inside of it
- Multiple paths from storage to servers
  - This is hard with embedded
  - Unless the product does it for you!
- Every OSS should be the same
  - Spare pool should match too!

Specific Design Points

- Understand your transport
  - We're pretty good with IB and Seastar/Gemini
  - Not great at TCP LND
- Configuration Management
  - Every node gets the same specification
- Centralized Syslog
  - Notifications from parsing

Case Study #1

- DDN 9900 storage system
  - 5 disk enclosures, 2 controllers, 4 OSS nodes
- Disk replacement around noon
- IOM failure around 1:30am
- Does anyone see the problem?

Case Study #1

- If the IOM had failed in the same enclosure as the rebuilding disk we're in the clear
- …but it didn't.
- Controller keeps a journal to replay, but it's not persistent across reboots.
- But the only thing that made sense at the time is to reboot the controllers
  - They don't really report failed IOMs well

Case Study #1

- Now we're in real trouble.
- Mistake #??
  - We didn't involve our Lustre support team soon enough.
  - Engaged HW support early on (++good)
- Where do we go from here?
  - Job scheduling paused for ~12 hours
  - Interactive logins are having problems
  - Tickets rolling in to help desk
  - Management getting calls from program managers…

Moving forward…

- Remove the OST from the filesystem so users stop getting IO hangs and only get IO errors
  - lctl conf_param {OST name}.osc.active=0
- While debugging the storage issue we ran some e2fsck's
  - Allowed it to make some changes
  - CTRL-C'd it about 100k changes in
  - So we're in a real pickle!

Moving forward…

- Uncovered a bug in e2fsck
  - Got that fixed, but how to test?
- Luckily we had some storage laying around
  - Exact HW and configuration of problem HW
- Write a simple netcat server/client program to dd the data off the bad OST onto 21 separate targets
  - Planned on things not going well the first time
- While this is going on we use lfs find --obd
  - Took 6 days to list the files on this OST
  - ~1 million files

Moving forward…

- Several runs of e2fsck later we think things will work.
  - But we have pristine copies of the data to put back in place if it doesn't!
- Run e2fsck; cross fingers; close eyes…
- And it worked!
- Sort of..

The next problem

- The underlying filesytem (ext3/ldiskfs) is in order.
- The upper level filesystem has problems
  - Data blocks are not linked to the bitmaps anymore
- Re-enable the OST
  - lctl set_param osc.<fsname>-<OST name>-*.active=1

The next problem

- We have the listing of files on the OST
- Now we need to figure out if the file(s) touched that OST
  - Lots of files have less than full stripe count
- Found that ~900k files didn't actually touch that OST
  - Another way to read that - ~90% of files that were part of the 4 OST stripe were less than 4MB
- Now turn focus to the remaining files

We're almost there!

- Timeline: We're at day 6
- All files that had data on the OST were read using dd
- If file returned IO error, it was removed with unlink
  - rm calls stat; stat call fails on IO error..
- Send list of files affected to the users so they know what we deleted.
- Write after-action report and meet with managers
- Sleep…

Case Study #1 - Summary

- “triple disk failure”
- ~1.1M files affected
- 7 days of downtime for that OST
- 50k files deleted because they were damaged
- List of Lessons Learned
  - Always engage HW support before rebooting anything.
  - Engage SW support quickly
  - e2fsck –fy is **not** your friend
  - Go the extra expense, 10 trays for safety

Case Study #2

- How upgrades can go horribly, horribly wrong
- Background
- Lustre 2.4 Servers; Lustre 1.8.6 clients
- The plan: Upgrade clients to Lustre 2.4.0
- The problem: synthetic benchmarks don't mirror user activity and our biggest problem doesn't come close to meeting the users ability

Case Study #2

- For ~4 months take test shots with all of Titan using home-built Lustre 2.4.0 client
  - Some patching, not much
- Benchmarks
  - Mdtest, IOR, S3D
- So we had a user application in there!
  - We're not users or domain experts
  - We weren't running it right

Cronology

- September 30, 2013, Put 2.4.0 (with patches) FS into production with 1.8.6 clients and some 2.4.0 clients
- January 7, 2014 FS is made primary, existing 1.8.6 is marked as read-only
- January 10 – first report of “slowness” comes in
- January 28 – 1.8.6 FS is removed from all compute platforms.
- January 30, first downtime because MDS became unresponsive

Trust your monitoring

- Extremely high MDS loads began shortly after 1.8.6 FS went read only.
  - We increased the Nagios check threshold
  - Figured increased cores on the MDS, needed to change the value
  - We were wrong
- MDSTrace wasn't helping
  - Only saw normal activity
  - We missed some new RPC's for Lustre 2.X

The problem

- Wide Striping
  - A feature we paid for
  - A feature we ended up developing and testing in-house
  - A feature we merged into the mainline code
  - A feature we failed to test during our 4 months of testing
- Most of the testing was done on a filesystem that didn't have more than 500 OST's
  - Seems to be >680 OSTs where we start to see the problem.

vmalloc vs. kmalloc

- kmalloc requests contiguous memory only, fails if the size you request isn't available.
- vmalloc uses the virtual memory maps in the kernel to grant non-contiguous pages that look like contiguous pages.
- Since vmalloc uses the virtual memory maps, it uses a global spin lock to keep operations atomic
- Lustre defines the break point of 16k to be when it requests either kmalloc or vmalloc

What does this have to do with anything?

- MDS reply buffer allocated based on the number of OST's in the file
  - But default behavior is to be aggressive and not need to re-request or expand the size of a buffer
  - So assume the largest buffer possible based on Lustre config
- 32 byte header plus 24 bytes * # of OST's
- Cross the 16k threshold at 680 OST
- Production filesystems at OLCF have 1008 OST's

How did we figure this out?

- We engaged our support vendor who sent on-site staff
- We got all of our hands on deck
- We all met in the same room and debugged
- Our developer and their developer started talking
- A few key observations made
- Patch was written and tested on test system then sent to Gerrit

LU-4008

- Every stat call was generating a vmalloc.
- ls –l in a directory of 50,000 files generates 50,000 vmalloc's
- Even if the file is only striped across 4 OST's
- Don't see this if filesystem has less than 680 OST
  - Our testing configuration didn't
- Detailed analysis showed several calls
  - getattr, layout lock intent, LOV_EA reply buffer

LU-4008 implementation

- Applied 2 patches (find them in LU-4008)
- Tested with a few user codes, put in production for bake
- Start looking at our testing rigor
- Why don’t we have real apps in our IO testing?
- We’re not domain scientists, don’t know how to run the app, don’t know what a good problem size is
- Don’t know what mix of apps to run in a diverse workload (production) simulation

IO workload harness

- Paper being presented at CUG in April
- Not going to steal their thunder
- We use this workload every time we test a patch or new version of Lustre
- Compare application runtime and IO performance of each app across runs to look for deviations
- Has been a very big asset
- Took ~1 year to build

Case Study #2 - Summary

- Locking in Lustre is hard
- Scale exacerbates problems that may not exist at smaller scale
- Monitoring is good, but need to be sure you're getting the whole picture
- Sometimes it takes getting all the right people in the room while the problem is happening
- Building an IO harness is very site specific but extremely useful

Summary

- Handle failures without looking like a chicken with your head cut off
- Take the time to fully understand the problem
- Where possible design (and fund!) around lessons learned
- Make sure your 8am brain is on
- Open communication with your users and management is essential

Questions?