Exascale: A Long Look at Lustre Limitations

LAD 2014
nathan.rutman@seagate.com

Agenda

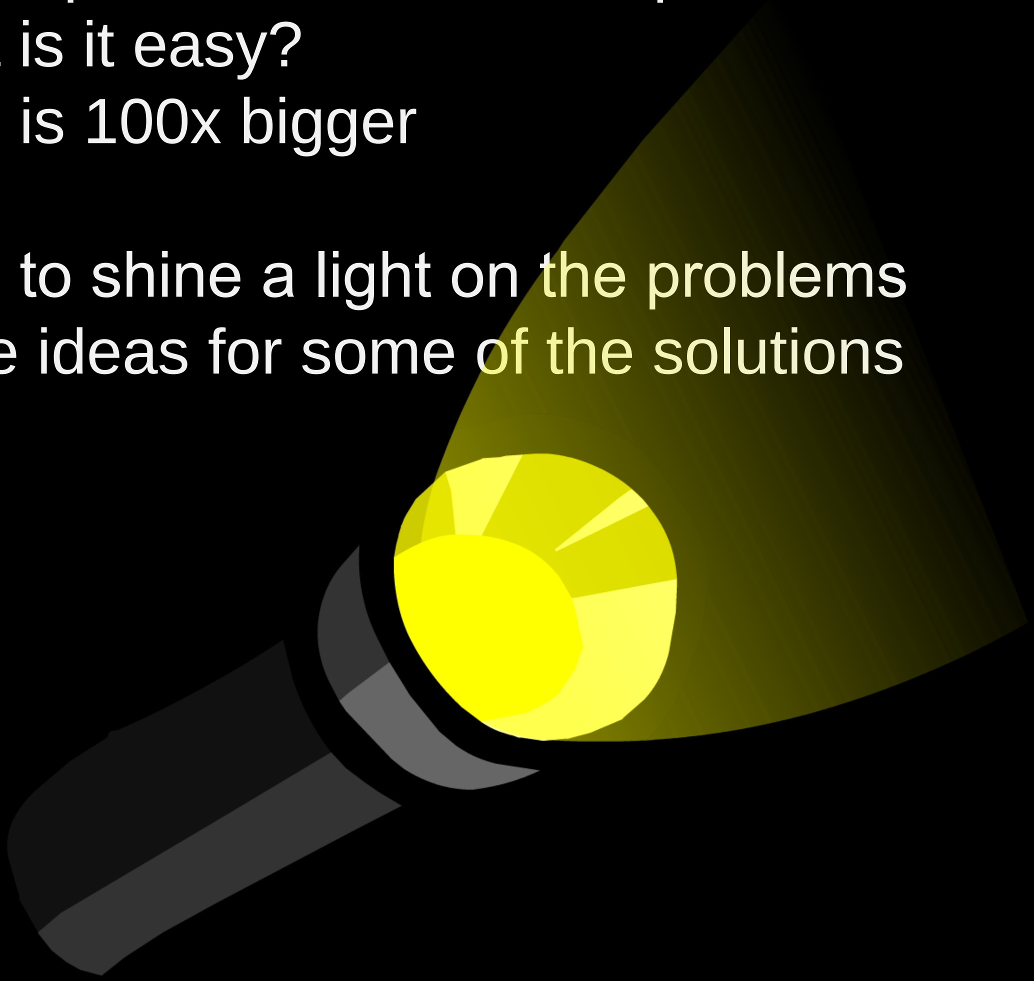
- What’s with your title?
- Lustre scale today
- Exascale differences
- Recovery
- Availability
- Network
- Hardware factors
- Layering
- Visibility
- Code Quality

What’s with your title?

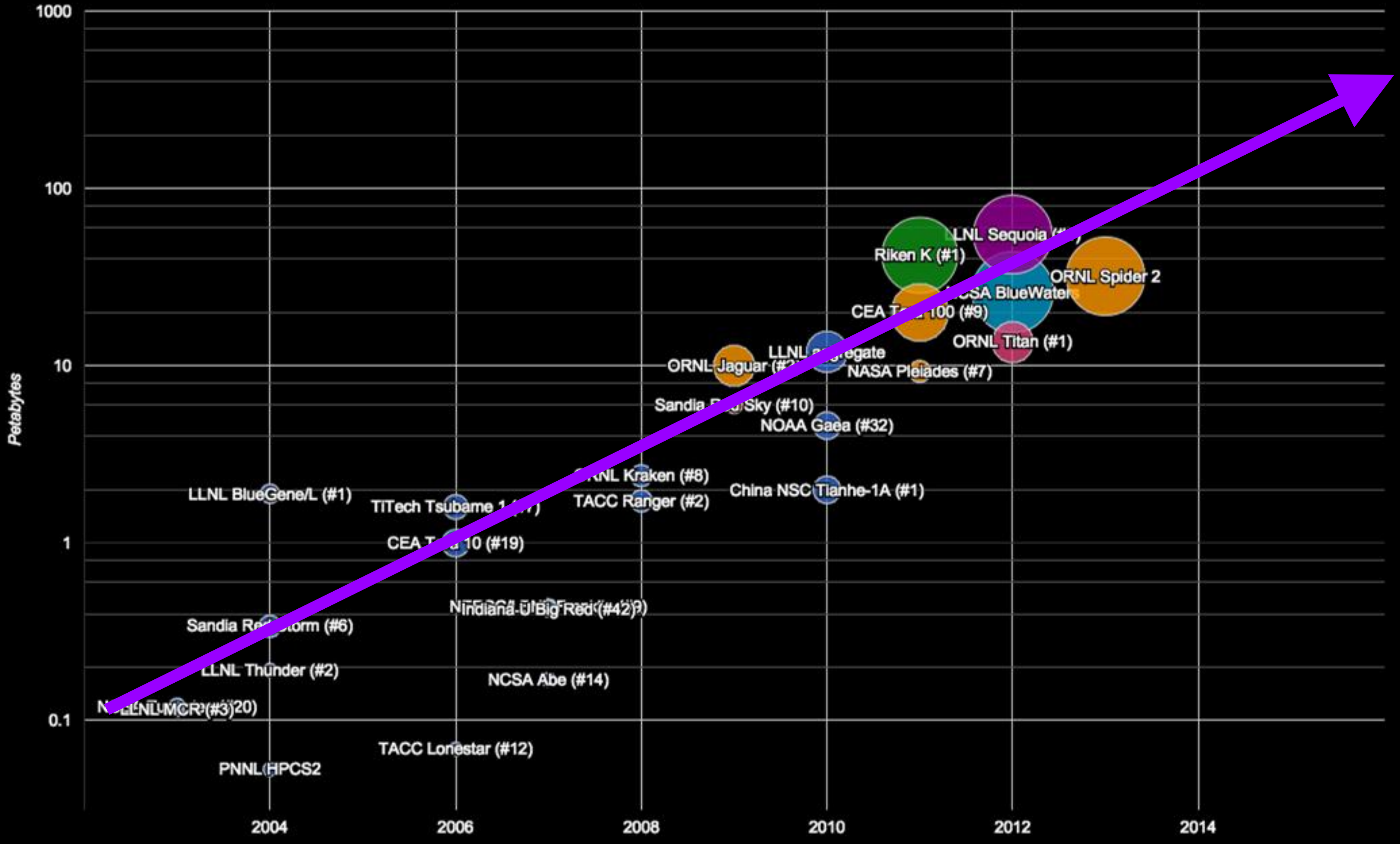
- Lustre is the biggest, baddest FS there is!
- 7+ of the top 10, tens of PB, TB per second
- Yes. But is it easy?
- Exascale is 100x bigger

- I’m going to shine a light on the problems
- There are ideas for some of the solutions

- but not all



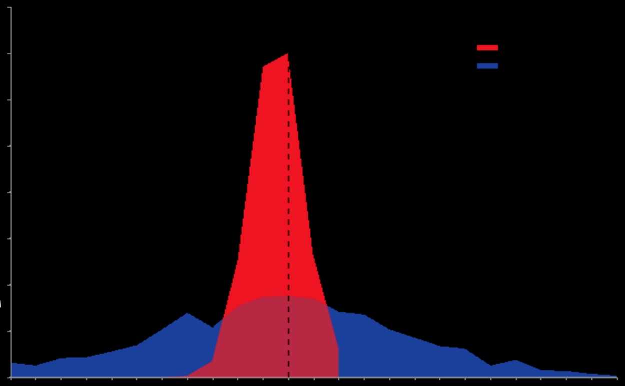
Lustre systems growth



Exascale differences

- Hardware scaling
  - Component Failures
  - Timeouts
  - Network losses
  - Hardware diversity
- Software scaling
  - Corner cases
  - Stack growth
- Complexity
  - Component count
  - Layer count
  - Cascading events
  - What’s going on?!?

Recovery



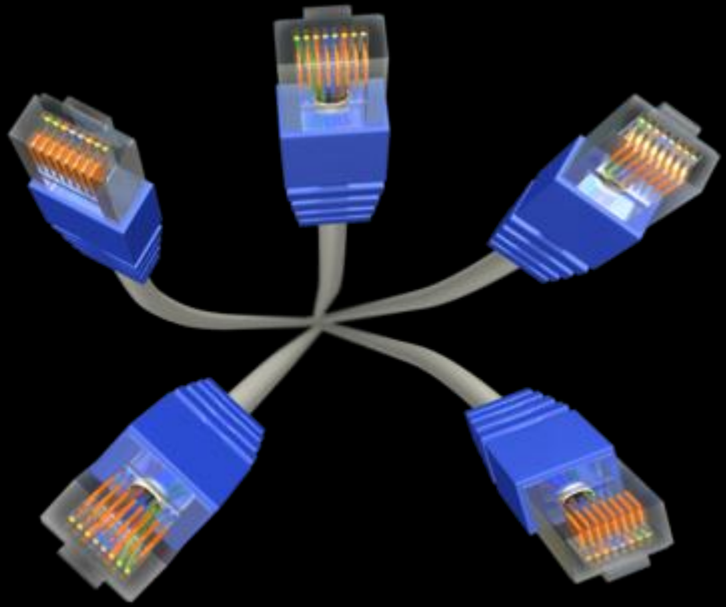
- Timeouts must increase with scale
    - must cover the worst case!
    - adaptive timeouts help to find the limits, but don’t change them
    - temporary outages - “beer timeouts”
- Recovery actions tied to timeouts
    - imperative recovery helps during failover
    - expected wait times for resend, lock callback, etc grow
- More components = more failures
    - drive failure
    - server failure
    - network packet loss
- More failures + longer recovery = not good

Availability

- At scale, there will always be an OST down
- Well, we’ve only lost access to some of our files…
- Fewer, bigger OSTs - ZFS?
  - Larger chance of OST rebuild
  - This is vertical, not horizontal scaling
- Fancier layouts - RAID1 too expensive, need RAID6
- Need to handle more than a few 1000 OSTs

Network

- LNET message queues are FIFO
  - actionable reqs stuck behind waiting ones
- Need channels with independent credits
- Need to figure out prioritization
- Unbelievably, still 1:1 client-server pinging
- Lustre is not robust in the face of dropped packets



Hardware Diversity

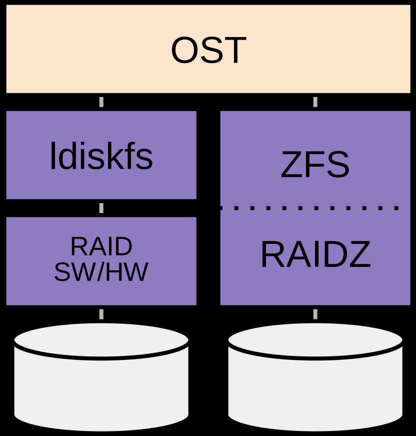
- Storage != Spinning discs
  - media hierarchy from RAM, NVRAM, disc, tape
- No in-Lustre hierarchy
  - need more descriptive layouts
    - extent-based current & goal
  - should handle more media types
  - automatic migration
- Client-server model
  - Can’t use storage on compute nodes
  - All resources managed by server - locks, grant, quota
  - No proxies - no localized caches
  - Converged client - Lustre 2.0

Server Hardware

- Cores and threads
  - what’s the right number?
  - big servers have thousands of threads - but most are just waiting
  - when requests > threads, they wait even though progress is possible
    - HPQ code is imperfect
    - timed-out client can't reconnect to release lock (LU-1239)
    - all-threads-busy scenarios are not well tested
- Sleeping hurts
  - cache line flush
  - paging
- Replace thread-per-req with cpu-localized state machines

Software Stack

- Parallel file system built on local filesystem
  - Allocator, elevator, request ordering, ldiskfs
  - RAID reordering
  - Interface limits efficiency: caching, readahead
  - Direct OSD devices?



- No hierarchy in Lustre for data movement
- Add hierarchy outsid
  - PLFS, Burst Buffer, I
  - Integration effort
  - Recovery / transacti
  - Who to blame?

Visibility

- Everybody loves syslog debugging
- Especially correlating across multiple nodes
  - Just collecting logs is a pain
- Kernel dumps and system panics are fun!
- Neither human- nor machine-readable
- Turn up debug level -- *after* you see the problem
- Need full-time, machine-readable, centrally collected debug data

```
00000100:00000001:6.0:1407191985.455969:0:19286:0:(client.c:1489:ptlrpc_check_set()) Process entered
00000400:00000001:6.0:1407191985.455971:0:19286:0:(lib-msg.c:48:lnet_build_unlink_event()) Process entered
00000400:00000001:6.0:1407191985.455972:0:19286:0:(lib-msg.c:57:lnet_build_unlink_event()) Process leaving
00000100:00000001:6.0:1407191985.455973:0:19286:0:(events.c:96:reply_in_callback()) Process entered
00000100:00000200:6.0:1407191985.455975:0:19286:0:(events.c:98:reply_in_callback()) @@@ type 6, status 0  req@ffff880835b04c00 x1475091387459632/t0(
ffff880839f80000@10.149.150.29@o2ib4010:28/4 lens 328/400 e 6 to 0 dl 1407192183 ref 1 fl Rpc:
RU/2/ffffffff rc -11/-1
00000100:00000200:6.0:1407191985.455981:0:19286:0:(events.c:119:reply_in_callback()) @@@ unlink  req@ffff880835b04c00 x1475091387459632/t0(0) o101-
ffff880839f80000@10.149.150.29@o2ib4010:28/4 lens 328/400 e 6 to 0 dl 1407192183 ref 1 fl Rpc:R/2/fffff
fff rc -11/-1
00000100:00000001:6.0:1407191985.455984:0:19286:0:(events.c:174:reply_in_callback()) Process leaving
00000400:00000200:6.0:1407191985.455985:0:19286:0:(lib-md.c:73:lnet_md_unlink()) Unlinking md ffff88075e926640
00000100:00000001:6.0:1407191985.455986:0:19286:0:(client.c:2353:ptlrpc_unregister_reply()) Process leaving (rc=1 : 1 : 1)
00000100:00000001:6.0:1407191985.455987:0:19286:0:(client.c:1194:after_reply()) Process entered
00000100:00000001:6.0:1407191985.455973:0:19286:0:(events.c:96:reply_in_callback()) Process entered
00000100:00000200:6.0:1407191985.455975:0:19286:0:(events.c:98:reply_in_callback()) @@@ type 6, status 0  req@ffff880835b04c00 x1475091387459632/t0(
ffff880839f80000@10.149.150.29@o2ib4010:28/4 lens 328/400 e 6 to 0 dl 1407192183 ref 1 fl Rpc:
RU/2/ffffffff rc -11/-1
00000100:00000200:6.0:1407191985.455981:0:19286:0:(events.c:119:reply_in_callback()) @@@ unlink  req@ffff880835b04c00 x1475091387459632/t0(0) o101-
ffff880839f80000@10.149.150.29@o2ib4010:28/4 lens 328/400 e 6 to 0 dl 1407192183 ref 1 fl Rpc:R/2/ffffffff rc -11/-1
00000100:00000001:6.0:1407191985.455984:0:19286:0:(events.c:174:reply_in_callback()) Process leaving
00000400:00000200:6.0:1407191985.455985:0:19286:0:(lib-md.c:73:lnet_md_unlink()) Unlinking md ffff88075e926640
00000100:00000001:6.0:1407191985.455986:0:19286:0:(client.c:2353:ptlrpc_unregister_reply()) Process leaving (rc=1 : 1 : 1)
00000100:00000001:6.0:1407191985.455987:0:19286:0:(client.c:1194:after_reply()) Process entered
02000000:00000001:6.0:1407191985.455988:0:19286:0:(sec.c:992:do_cli_unwrap_reply()) Process entered
02000000:00000001:6.0:1407191985.455988:0:19286:0:(sec.c:992:do_cli_unwrap_reply()) Process entered
00000100:00000001:6.0:1407191985.455989:0:19286:0:(pack_generic.c:580:__lustre_unpack_msg()) Process entered
00000100:00000001:6.0:1407191985.455990:0:19286:0:(pack_generic.c:599:__lustre_unpack_msg()) Process leaving (rc=0 : 0 : 0)
02000000:00000001:6.0:1407191985.455991:0:19286:0:(sec.c:1046:do_cli_unwrap_reply()) Process leaving (rc=0 : 0 : 0)
00000100:00000400:6.0:1407191985.455993:0:19286:0:(client.c:303:ptlrpc_at_adj_net_latency()) Reported service time 192 > total measured time 103
00000100:00000001:6.0:1407191985.475626:0:19286:0:(client.c:1131:ptlrpc_check_status()) Process entered
00000100:00000001:6.0:1407191985.475627:0:19286:0:(client.c:1154:ptlrpc_check_status()) Process leaving (rc=18446744073709551605 : -11 : fffffffffffffff5)
00000100:00000001:6.0:1407191985.475628:0:19286:0:(client.c:2410:ptlrpc_free_committed()) Process entered
```

HA

- HA is a separate system
- Only a gross interaction of “failover” or not
- Network partition = evict all clients
- Need state knowledge *before* sending req/timeout
- Should incorporate external knowledge of cluster state
    - Clients
    - Network
- Node death on Lustre SW failure makes recovery actions more difficult
- Dual-ported drives risk user/admin/HA corruption

Lustre Code

- Lustre designed in 1999, for Petascale
- Lots of revision over time
- Explosion in complexity
- Changes often have unforeseen consequences
- Nobody has a full view anymore
- Poorly documented
- Cruft on cruft

Wednesday, 02 June, 1999 19:50:53

pschwan

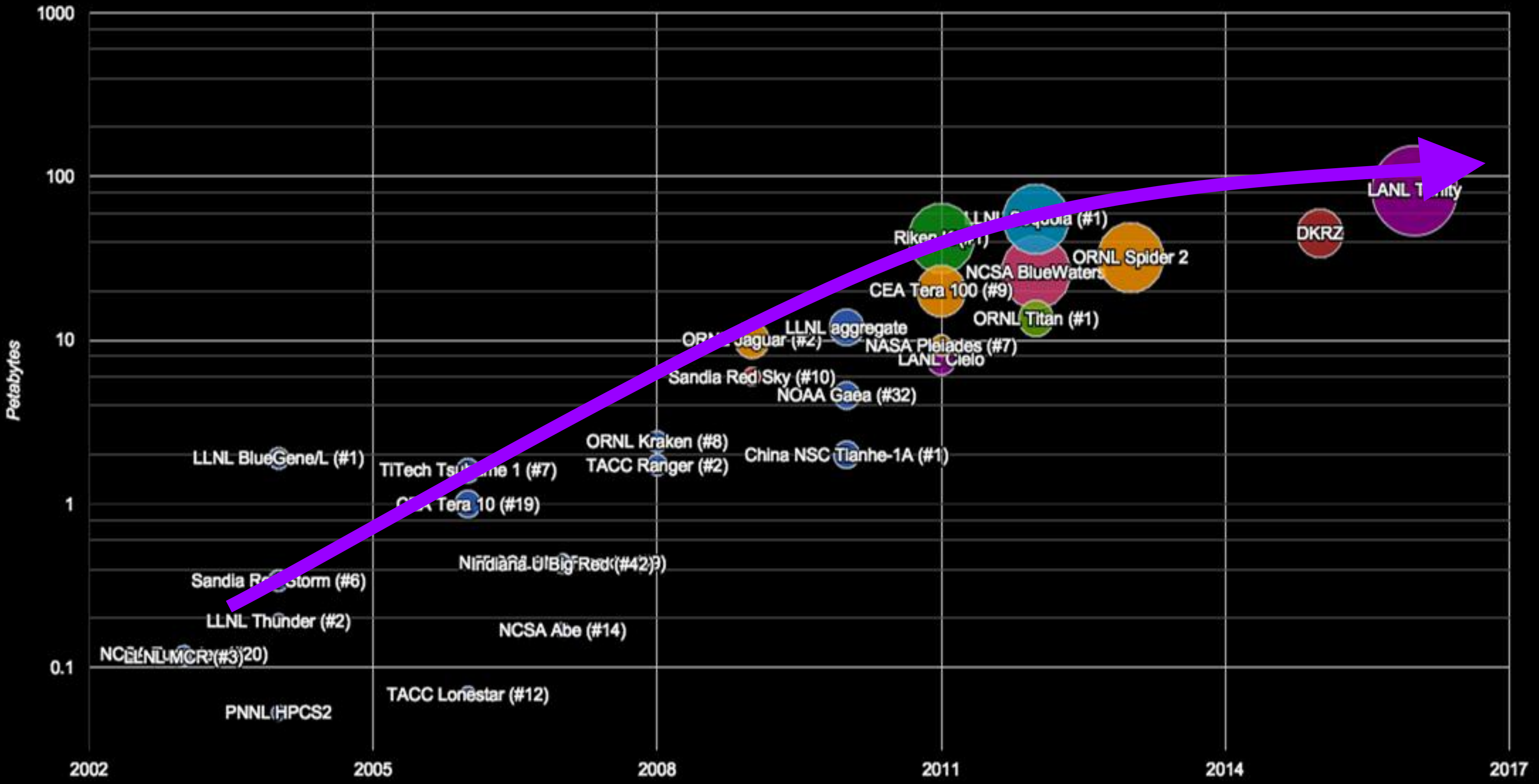
What are you doing about it?

- The problems are substantial
- We are working mainly to stabilize Lustre for current scale customers
  - RPC queues
  - flock scaling
  - hardening Recovery
  - lost packets
- But this in a sense is only fixing symptoms of the foundational problems
- Have we reached the saturation point with Lustre scale?

Thanks!

nathan.rutman@seagate.com

Lustre systems growth


Petabytes
LANL Trinity
DKRZ
LLNL Sequoia (#1)
Riken (#1)
ORNL Spider 2
NCSA BlueWaters
CEA Tera 100 (#9)
ORNL Titan (#1)
LLNL aggregate
ORNL Jaguar (#2)
NASA Pleiades (#7)
LANL Cielo
Sandia Red Sky (#10)
NOAA Gaea (#32)
ORNL Kraken (#8)
TACC Ranger (#2)
China NSC Tianhe-1A (#1)
LLNL BlueGene/L (#1)
TiTech Tsubame 1 (#7)
CEA Tera 10 (#19)
Indiana U Big Red (#42)
Sandia Red Storm (#6)
LLNL Thunder (#2)
NCSA Abe (#14)
LLNL MCR (#3)
(#20)
TACC Lonestar (#12)
PNNL HPCS2