In-board Lustre on a Performance Tier

LAD 2017
Cory Spitz and Patrick Farrell
Cray Inc.

Abstract

Lustre is often deployed as HPC scratch in concert with a site-wide filesystem, but rarely is it deployed with flash. Possibly this is because of a common perception that Lustre cannot fully leverage the inherent capability of flash technology. We have prototyped a so-called in-board Lustre solution to asses use cases and performance issues with running Lustre on flash.

The in-board file system can be either part of the site-wide namespace or wholly separate. In the former case a policy engine can seamlessly rebalance Lustre storage targets. In the latter case, users must do explicit data movement between the Lustre instances to stage or persist their data, which can be done with `lfs migrate` or via a WLM.

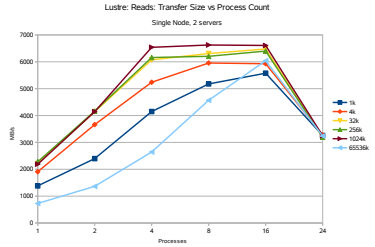
This talk will present the performance capabilities and the use cases for Lustre on a performance tier. It will also delve into future possibilities given the direction of the Lustre roadmap.

Lustre on flash as primary? Yes!

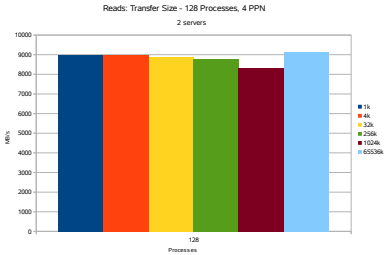
- It is possible to consider flash for primary storage
- Lustre on flash didn't always make sense
  - SW: mantra has been, "don't do small I/O on Lustre"
  - HW: didn't make sense either as disk can saturate network
- It does now!
- Latency and bandwidth of small I/Os is decent
  - See Patrick Farrell's upcoming LAD and Dev Summit talks
- Large bandwidth is exploitable with multi-rail LNet
- Life is good!

Lustre performs on flash

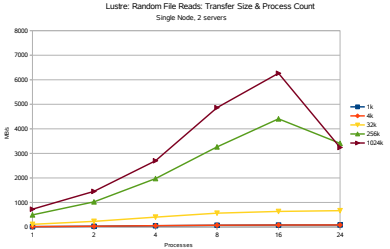
FPP Reads: Lustre



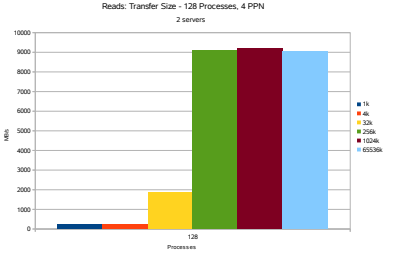
FPP Reads: Lustre – Many clients



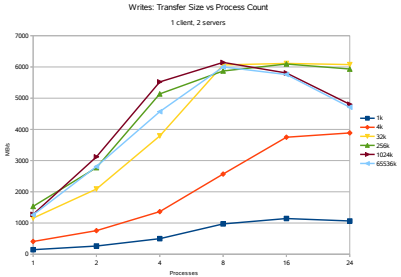
Random Reads: Lustre



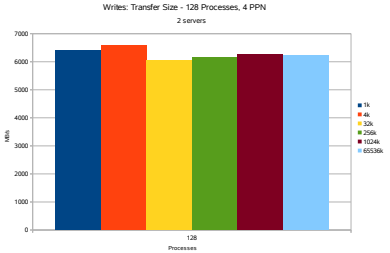
Random Reads: Lustre – Many clients



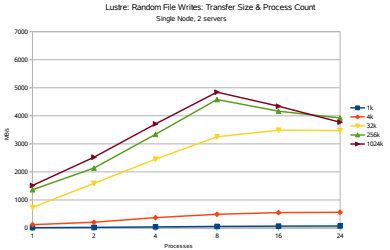
FPP Writes: Lustre



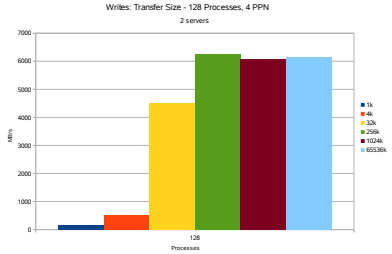
FPP Writes: Lustre – Many clients



Random Writes: Lustre



Random Writes: Lustre – Many clients

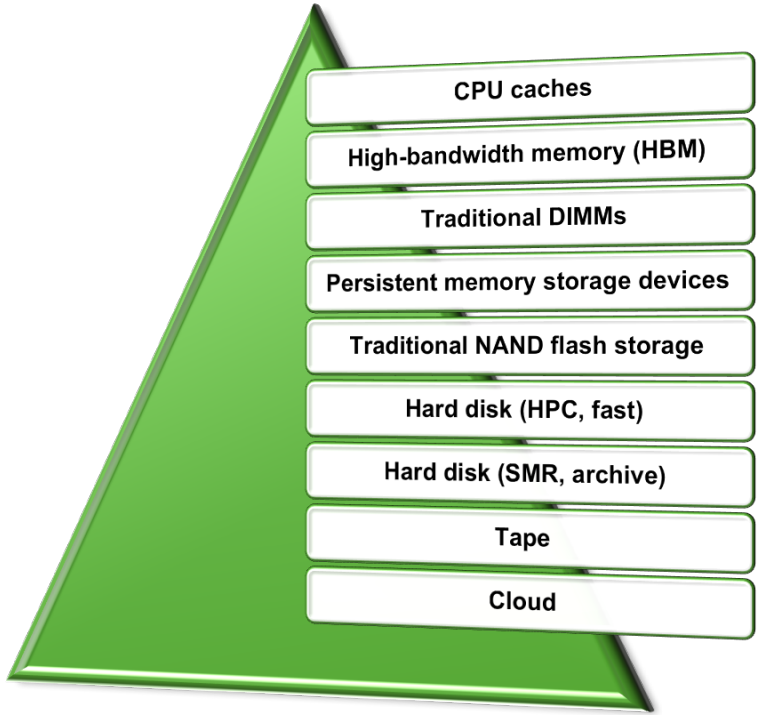


We need tiers and tiers are hard

- **Life is good! Right?**
  - Flash is too limited in capacity, still need a capacity tier
  - Conclusion: flash will augment disk
- **John Bent, 2017 ORNL Lustre Ecosystem Keynote:**
  Lang's Law – "The more tiers, the more tears"
  Bent – "the correct number of tiers is two"
- **Agreed, it's too hard to manage separate levels and namespaces**
- **Yet, we want Lustre on both flash and disk**

Consider the evolving memory-storage hierarchy

- There may be many physical layers
- Data must move between the layers
- Storage class memories are persistent, but can only serve as cache
- Flash can be private, shared, or embedded
- We want this to be as simple as possible
  - Ease of use
  - Ease of management



Add in-board Lustre performance tier

- **Bent didn't consider flash for Lustre itself**
  - We will soon see that the performance is enough
  - But we will need more than all-flash arrays
- **There are existing Lustre management tools to leverage**
  - OST pools naturally segregate storage classes
  - Complex layouts and progressive file layouts are a boon to tiers
    - Data on Metadata and performance flash alike
  - File level replication is coming soon
- **We can do DIY tiers, but tiers should be a 1st class Lustre citizen**
  - More implicit and explicit data placement is needed
  - Implicit and explicit data movement is needed
  - Data awareness for data management is needed

Lustre specific issues

- **It is too easy to fill a small flash OST**
  - Relatively small capacity and lots of demand
  - There is no way to restrict a tier/pool
  - Subdirectory mounts don’t help here

- **PFL is great until you fill the tier, then what?**
  - Users’ jobs fail; call operators
  - Operators step in and stop jobs
  - Use manual tools to copy data

How do you drain a tier?

- **How do you orchestrate and marshal data movement?**
  - Make the user do it or specify it? How?
  - Can the user control it? With an API?
  - Does the system only do it?
  - Or, can it be both?

- **Is it a burst buffer?**
  - Maybe you don't need to drain the whole thing
  - Lustre has no awareness of data's purpose

- **Does/must the user decide what to keep and what to leave?**
  - How can we empower the users or the operators to decide?

Draining logistical questions

- **Can the contents be tar'd, de-dup'd, etc.?**
  - We don't want to double the metadata load
  - Remember too that we are persisting to non-flash storage
  - Although, as Bent said, there are ways to augment disk transparently
- **Can the process start before the job is done?**
  - Before file close?
- **Can a tier appear to be an ideal data sink that never fills?**
  - And retain flash-like levels of performance?

Data placement questions

Today's PFS users are accustomed to dealing with data placement issues such as file/directory striping including stripe width and count

- **What control do we expect users to have with tiers?**
  - It seems that OST is the wrong granularity, pools are better
- **OSTs aren't treated a scarce resource today**
  - Continue with the free-for-all? Pool quotas?
- **In what way should the control be expressed?**
  - Absolute or relative size, physical location, or unexpressed

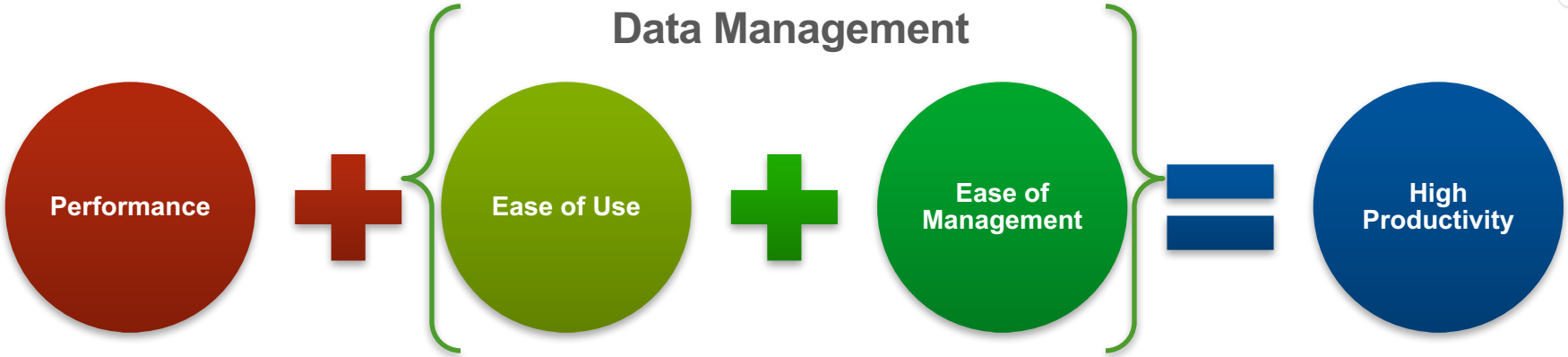
Lustre needs answers

- **The previous issues mostly boil down to:**
  - Data placement
  - Data movement
  - Data management
- **It seems clear that we need:**
  - Pool quotas, complex layouts, and FLR
  - Automation of data movement
    - Via policies
    - Controlled programmatically
    - Expressed implicitly and explicitly
  - Ease of management
    - Mostly external to Lustre; e.g. ladvise(), policy engine, or job schedulers
- **This is how we get from Lustre pools to tiers**

Ranked list of solutions

1. **Finish FLR phase I**
2. **Need an data orchestrator to initiate data movement**
   - External policy engine, job scheduler, or script
3. **Administrative tools and quotas for pools**
   - More easily done with extensions to project quotas
   - Seamlessly break out of a tier via PFL; first layout extent equals pool quota
4. **FLR copy while file is open for write**
   - Abort copy if overwritten
   - Later, consider partial HSM-archive and/or direct server copy
5. **Add Lustre clients to pool servers to efficiently push data out – maybe**
   - Bypass LNet and do efficient server side copies
   - This only makes good sense for single striped files
6. **Treat sets of small files within tiers as a group**
   - Tar up small singly striped files
   - Or use a loop-back file system
7. **Extend the reach (and performance) of flash tiers with client-side compression**

Data management yields high productivity



- **Performance – small file I/O, single client, streaming data**
  - Flash and SW improvements widen Lustre's sweetspot
- **Ease of Use – implicit and explicit automation of data movement**
- **Ease of Management – easy to administer and operate**

Merci!