LA-UR-19-24149

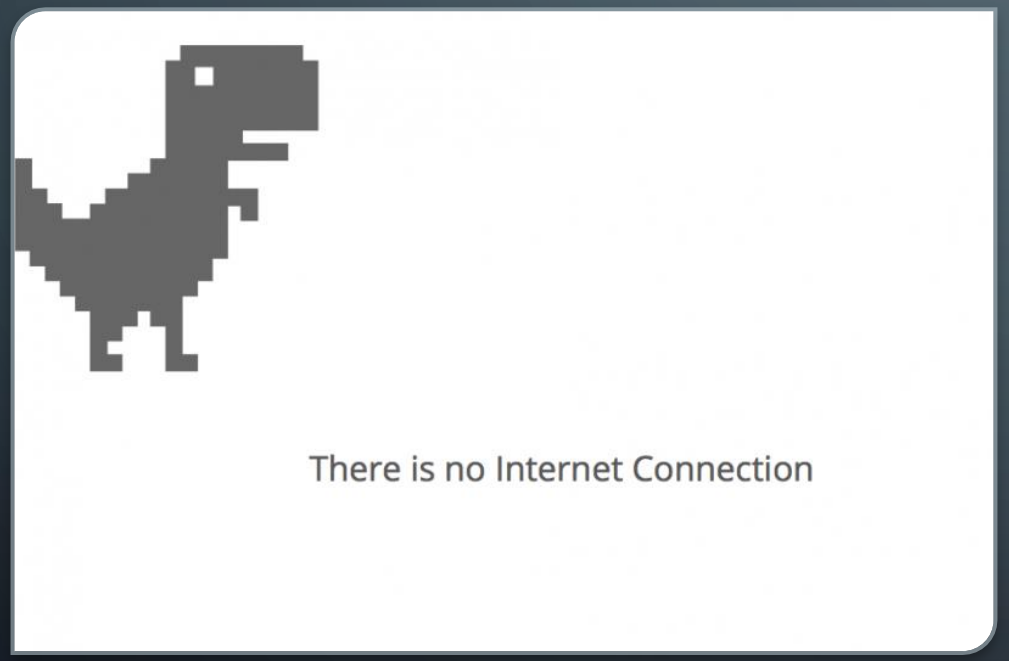
USERSPACE SQUASH FILESYSTEM FOR LAUNCHING LINUX CONTAINERS ON HPC SYSTEMS

BY RONALD SHANE GOFF

SOFTWARE IN HIGH PERFORMANCE COMPUTING

- Carefully administered and managed
  - Provides common tools needed by many such as MPI and compilers
- Only some versions of some things are installed
  - Might be missing an older or newer version you need
  - Can't install everything under the sun
- New software can be requested
  - Without great enough need the burden of support is too high

BUILDING YOUR OWN SOFTWARE



- You can build your own software for most things
- Can be a hassle to debug and repeat this
- Usually no internet access
- Have to rebuild every time on a new platform

THE QUEST FOR SOFTWARE ISOLATION

- Many people have specific dependencies that are not supported
- Building software can be difficult and can create conflicts with installed software (Priedhorsky & Randles, 2017)
- If you change platforms, you have to build everything again
- If you change software versions you have to build again
- Must get the software to the compute center without the internet

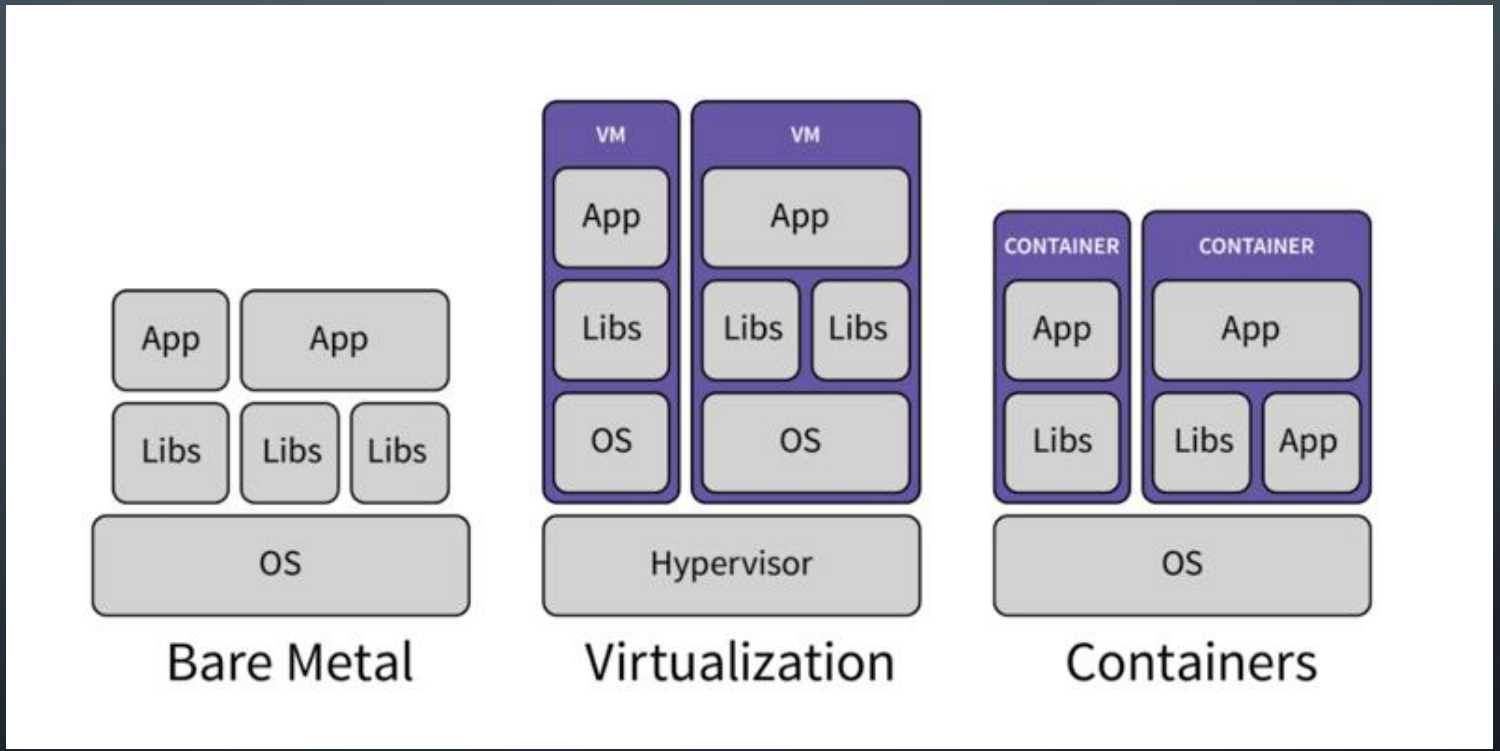
VIRTUAL MACHINES FOR ISOLATION

- Uses an entire guest operating system
- Separate user libraries and binaries from the host
- Separate kernel from the host
- Uses a hypervisor to give guest access to host hardware
- Almost entirely separate from the host
- Usually no access to special hardware used in HPC (Soltesz, Pötzl, Fiuczynski, Bavier, & Peterson, 2007)
- Keeps host kernel safe from blunders (Soltesz, Pötzl, Fiuczynski, Bavier, & Peterson, 2007)

LINUX CONTAINERS

- Separates user binaries and libraries from the host
- Uses Linux name spaces to separate resources instead of a hypervisor (Priedhorsky & Randles, 2017)
- Shares operating system kernel with the host
  - Can directly access special hardware with proper configuration (Priedhorsky & Randles, 2017)
  - Smaller than virtual machines (Jacobsen & Canon, 2015)
  - Can pose more risk to the host system, but can be mitigated (Jacobsen & Canon, 2015)
- Lower overhead than virtual machines(Jacobsen & Canon, 2015)

CONTAINERS ARE CLOSER TO BARE METAL



WHY CONTAINERS

- Fast (Jacobsen & Canon, 2015)
- Lightweight (Jacobsen & Canon, 2015)
- Unfettered internet access at build time
- Reproducible (Priedhorsky & Randles, 2017)
- Portable (Priedhorsky & Randles, 2017)

FITTING USER NEEDS

- Users can use the exact versions of everything they want
- Some users require validated software stacks to meet field specifications
- Some common software is still important to users, and can be used without having to build it all over again. They still have access to the managed stack
- Easy to experiment with new software

WE MUST ALSO ASK WHY NOT CONTAINERS

- User defined software stacks can introduce new attack vectors (Priedhorsky & Randles, 2017)
- Some container run time tools require root
- Containers can use more system memory than bare metal

UNPRIVILEGED WORKFLOW

- Security can be a priority
- Linux containers do not need root to execute (Priedhorsky & Randles, 2017)
- Some tools provide services to users that need root (Le & Paz, 2017)
  - Container build time requires root commonly
  - Uses a root-level process to start user containers without privilege
  - Parent process also handles root-level things it needs to work
  - Concerning to most cluster admins, can expose root access to users
  - Parent process with privilege is more dangerous than one without
  - Extra services mean extra effort to support

RUNNING CONTAINERS IN HPC

- Some tools require root-level access in some form for running containers
  - Docker swarm (Docker Engine)
  - Singularity (privileged parent process) https://singularity.lbl.gov/faq
  - Shifter (privileged parent process) https://github.com/NERSC/shifter/blob/master/doc/security.rst
- Some do not
  - Charliecloud

WHY CHARLIECLOUD

- Charliecloud does not require root
  - Does not use setuid
- It is developed at LANL
- It is open source
- It is light weight
  - It is only a container run time tool with some helper scripts
  - It provides tools to assist building containers, but does not build them itself

USING CHARLIECLOUD

- To use containers on HPC resources the container must be made available to all compute nodes within the cluster in some way
- Charliecloud only requires a directory tree resembling a Linux filesystem tree
- Docker containers are exported to a compressed tar file, then unpacked to memory on each node. This workflow is supported, but not required (which is cool).
- Provides tools for using some special HPC hardware (Priedhorsky & Randles, 2017)
- Uses host network stack instead of network namespaces (Priedhorsky & Randles, 2017)

OTHER WAYS TO DISTRIBUTE CONTAINERS

- You can run the container from a networked filesystem
  - No need to distribute containers manually
  - Job time vs run time
  - longer run time of containers
  - Only reads necessary files instead of the whole container, using less memory compared with copying the entire container to memory

ENTER THE SQUASH FILESYSTEM (SQUASHFS)

- Compressed read only filesystem
- Already used by some container tools, like Shifter
  - Uses a kernel mounted squash filesystem to make container images available over a network filesystem
- Kernel mounted is proven to be fast and efficient for containers in HPC (Gerhardt et al., 2017)
- Hosted on an existing network filesystem like Lustre (Gerhardt et al., 2017)

DO WE NEED ROOT FOR SQUASHFS?

- Kernel mounted squashfs is fast and great, but needs root
- Existing tools can make use of the squashfs without root via Filesystems in userspace (FUSE)
- FUSE is generally considered to add overhead (Vangoor, Tarasov, & Zadok, n.d.)

A LITTLE ABOUT LUSTRE

- Parallel filesystem that uses object storage
- Separates file data and metadata
  - Uses servers to make bits available
  - Metadata server
  - Metadata targets
  - Object Storage server
  - Object Storage targets

MORE ABOUT LUSTRE

- Lustre seperates data intro stripes for parallelism
- Lustre likes large contiguous parallel operations
- Lustre dislikes many small files
- Lustre performance suffers under heavy metadata loads

LUSTRE AND SQUASHFS

- Lustre doesn't deal well with small file transactions, due to having separate metadata (Gerhardt et al., 2017)
- Containers can have many small file transactions, as most binaries and libraries aren't very large
- Squashfs contains and compresses metadata
  - Keeps everything bundled together
  - Presented as a single file to Lustre, avoiding lots of metadata transactions (Gerhardt et al., 2017)

FUSE

- FUSE allows users to mount filesystems without root
- User level processes communicate with kernel driver to do work
  - Users can ONLY start user level processes, kernel driver is separate and handles root stuff
- Low-level FUSE API
  - Faster, harder to develop (Vangoor, Tarasov, & Zadok, n.d.)
- High-level FUSE API
  - Wraps around low-level API, adds more overhead converting paths to inodes (Vangoor, Tarasov, & Zadok, n.d.)

OBJECTIVES

- Find out what amount of overhead is added to squashfs through FUSE
- Explore options to minimize overhead, and optimize the squashfs on Lustre
  - Containers don't fit well in Lustre best practices
  - There are options for squashfs block size we can adjust
  - There are Lustre options we can adjust for squashfs

THE PYNAMIC BENCHMARK

- Developed at Lawrence Livermore National Laboratory
- Designed to test a systems ability to handle the Dynamic Linking and Loading requirements of Python-based scientific applications
- Creates a number of arbitrary library files
- Reports timings for loading these files

METHOD

Factors	Levels
Node Count	1,128,512,1024
Mount Method	kernel, squashfuse, squashfuse_ll
Squashfs Block size	128Kib, 1MiB
Lustre Stripe Size	64KiB, 1MiB
Lustre Stripe Pattern	1 OST, 2 OSTs, 32 OSTs

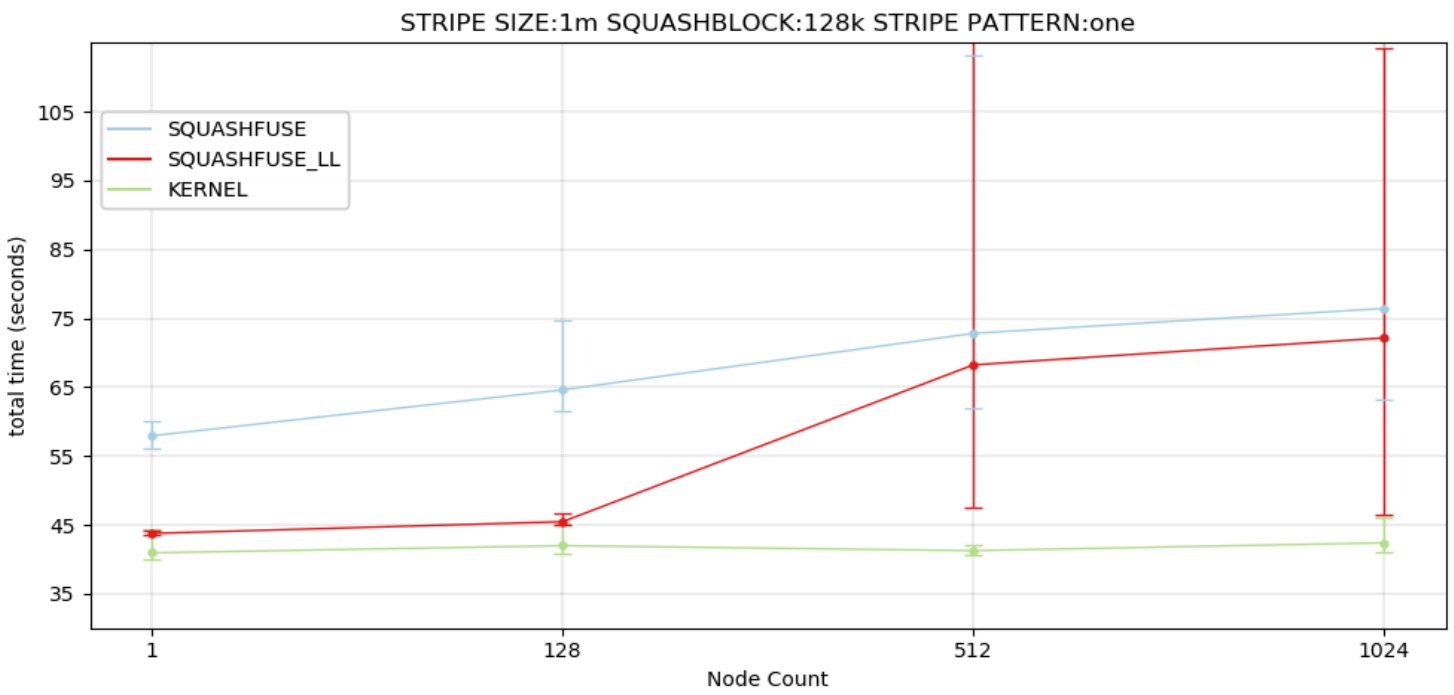
MISC

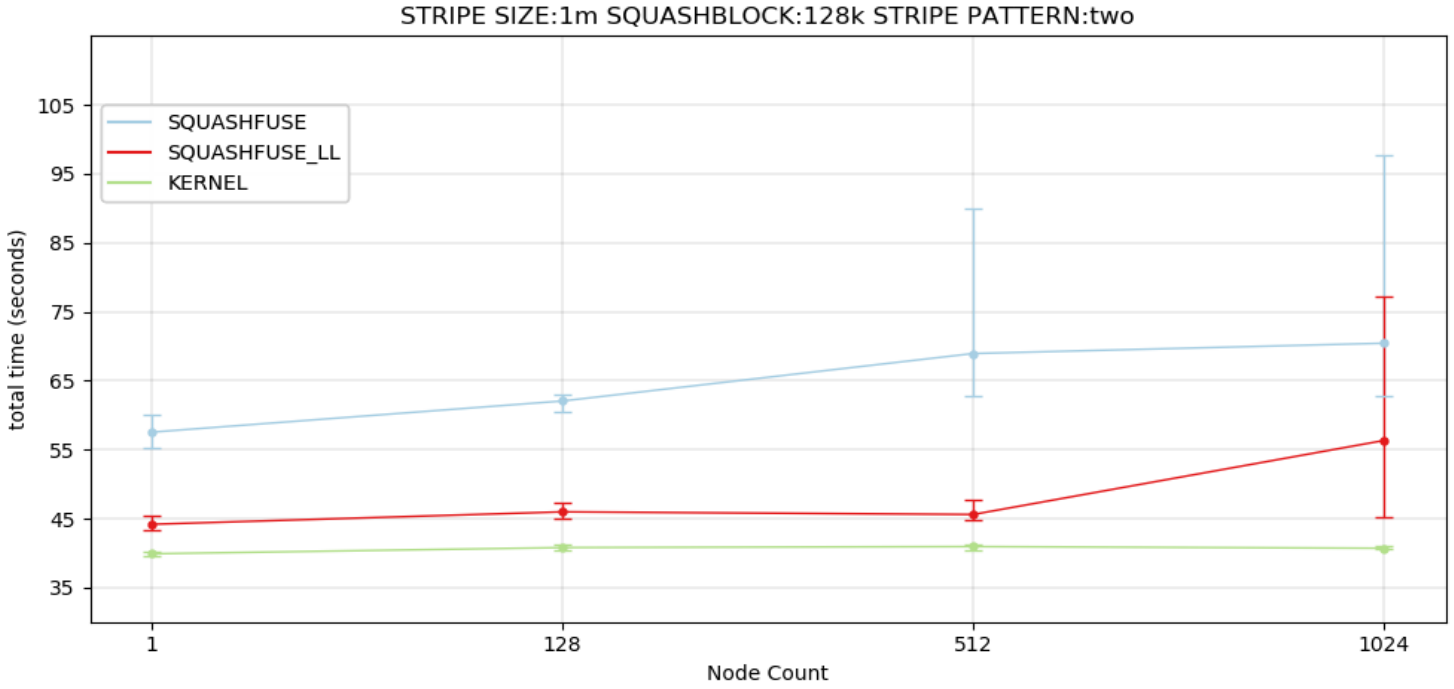
- Some experiment iterations were run on Network Filesystem instead of Lustre, for science.
- Not all things will resemble an HPC workflow
  - A recursive grep will be run on the squashfs which will walk the whole filesystem to test limits. Most scientific applications won't do such a thing, but someone will.

RESULTS

- Kernel mounted squashfs is king
  - Doesn't care about any attempts or lack of attempts at optimizing
  - Fastest in all configurations, very stable
- High-level FUSE API was slowest
  - Shows more impact regarding optimization
- Low-level FUSE API was in the middle
  - Shows similar but less obvious benefits from optimizations

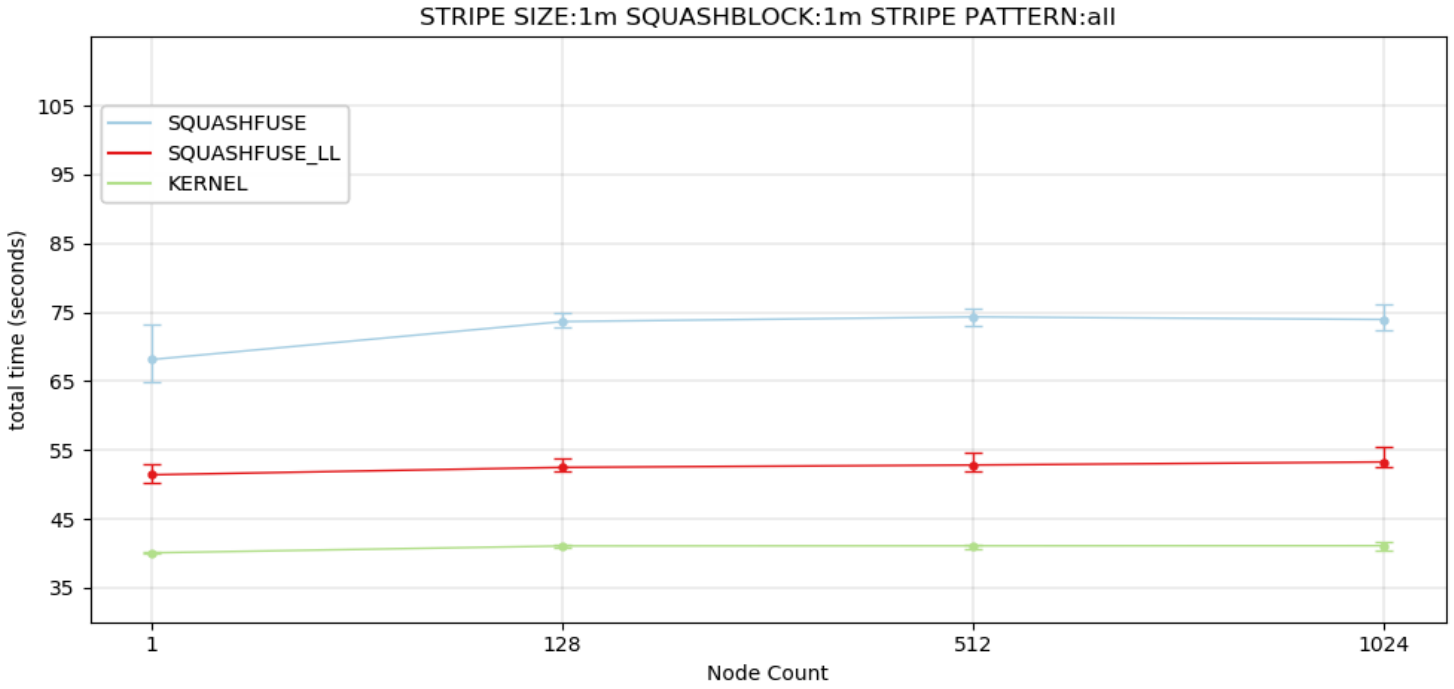
LUSTRE STRIPING MATTERS

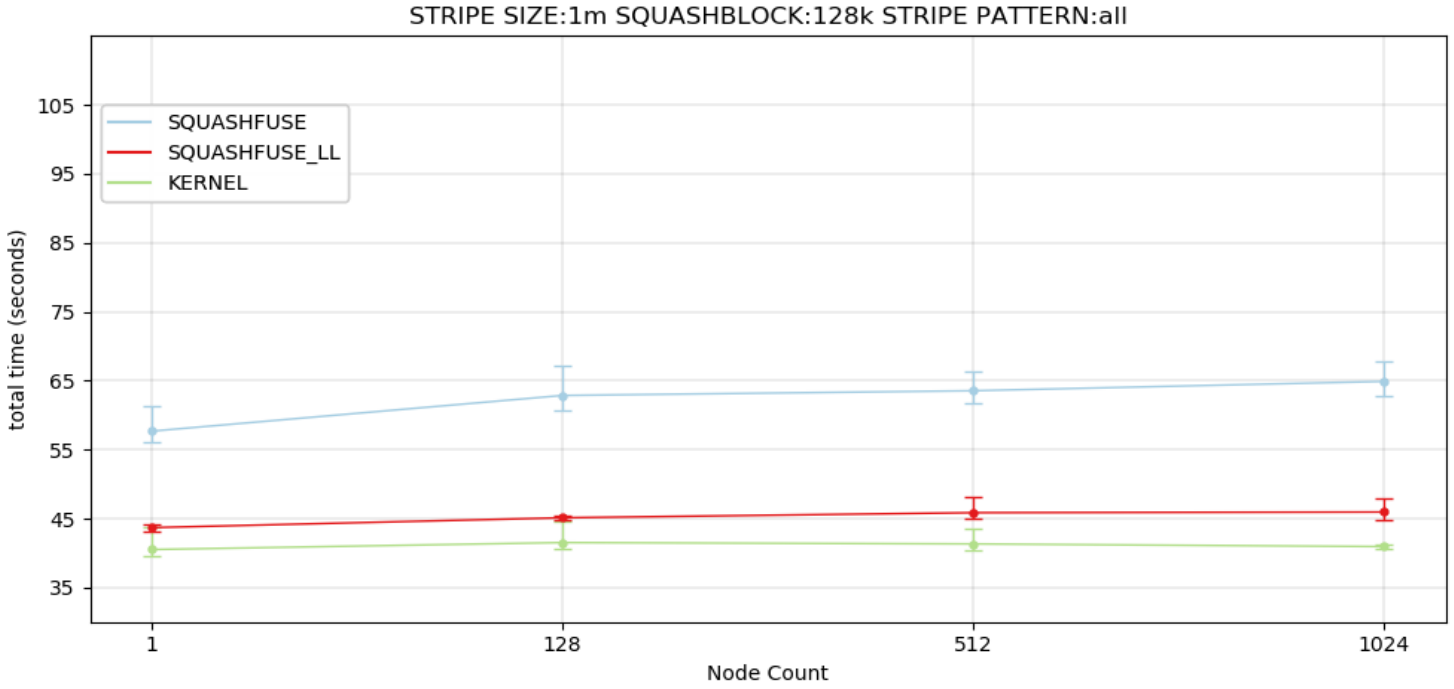




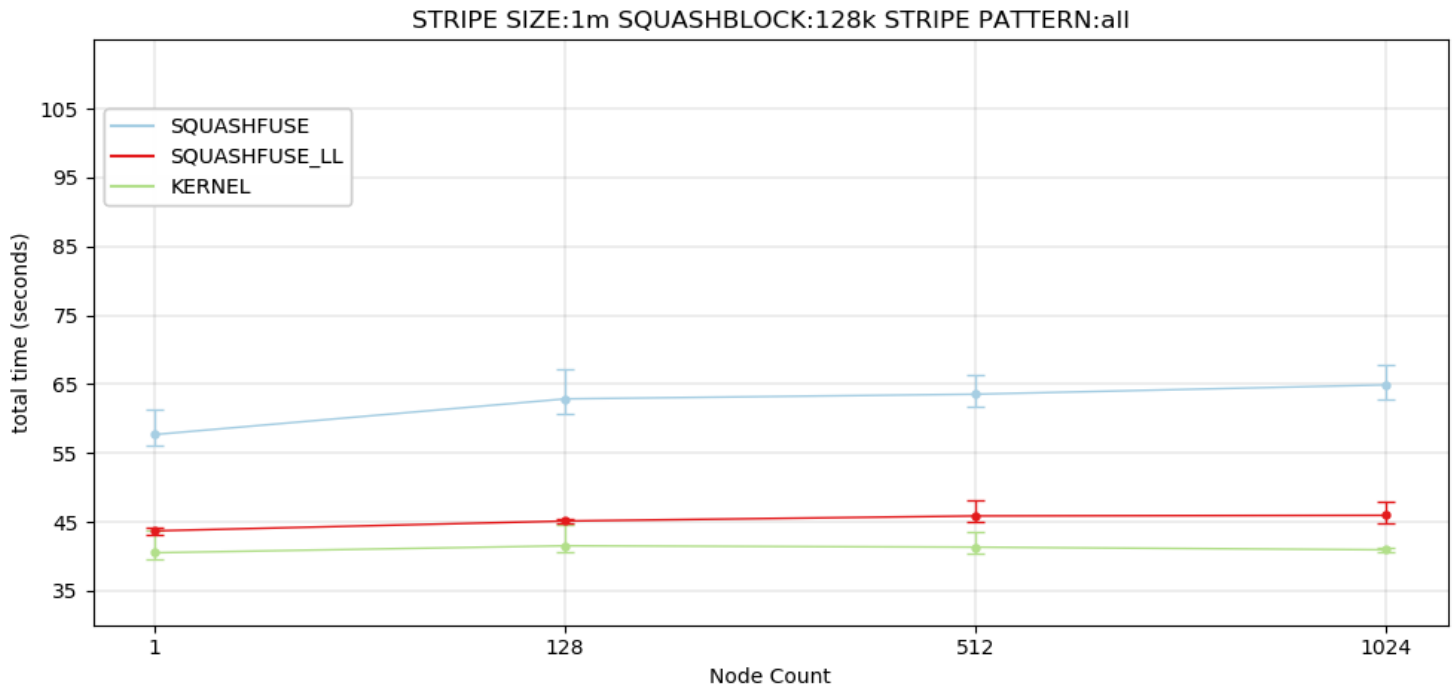
- Default squashfs block size, Lustre stripe size, and stripe count
- Increasing stripe pattern from one stripe, to two greatly improves performance as scale increases

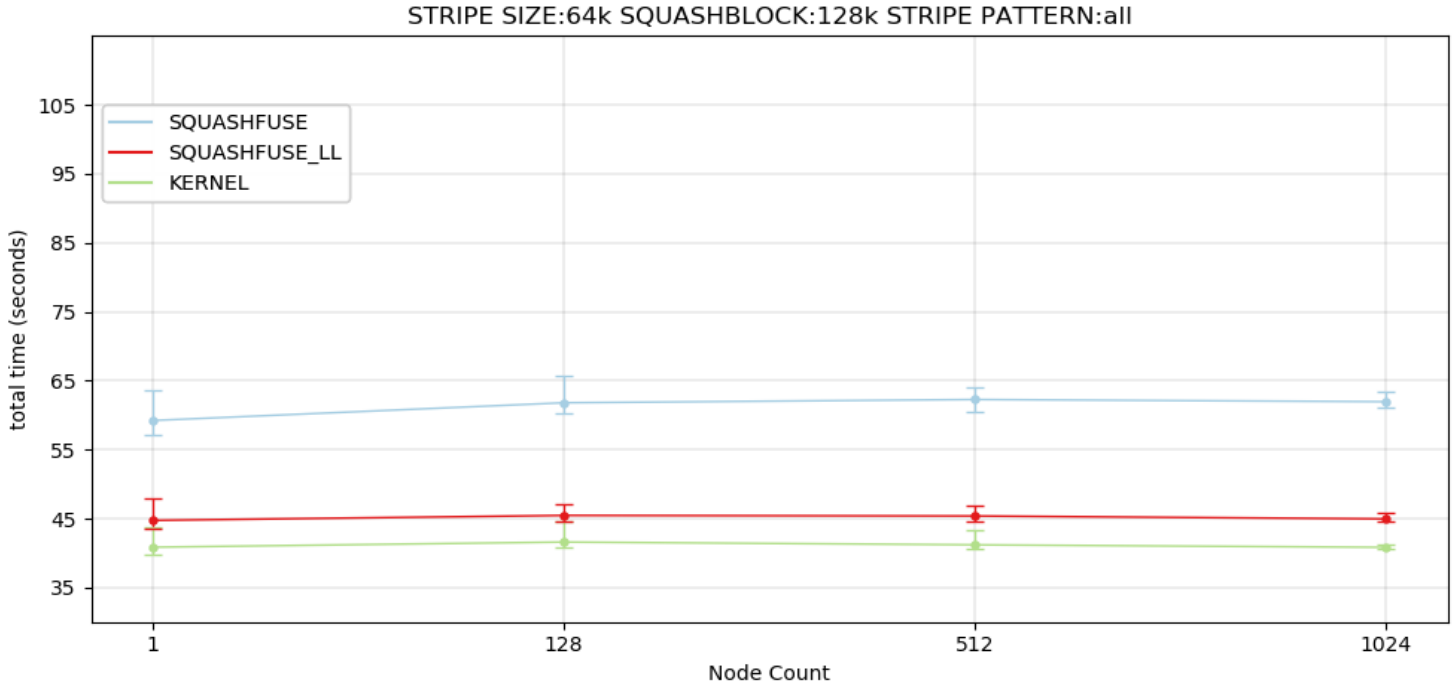
SQUASHFS BLOCK SIZE





- Smaller squashfs block size is faster than a larger one
- Squashfs block size can be smaller than the default 128KiB, but was not tested



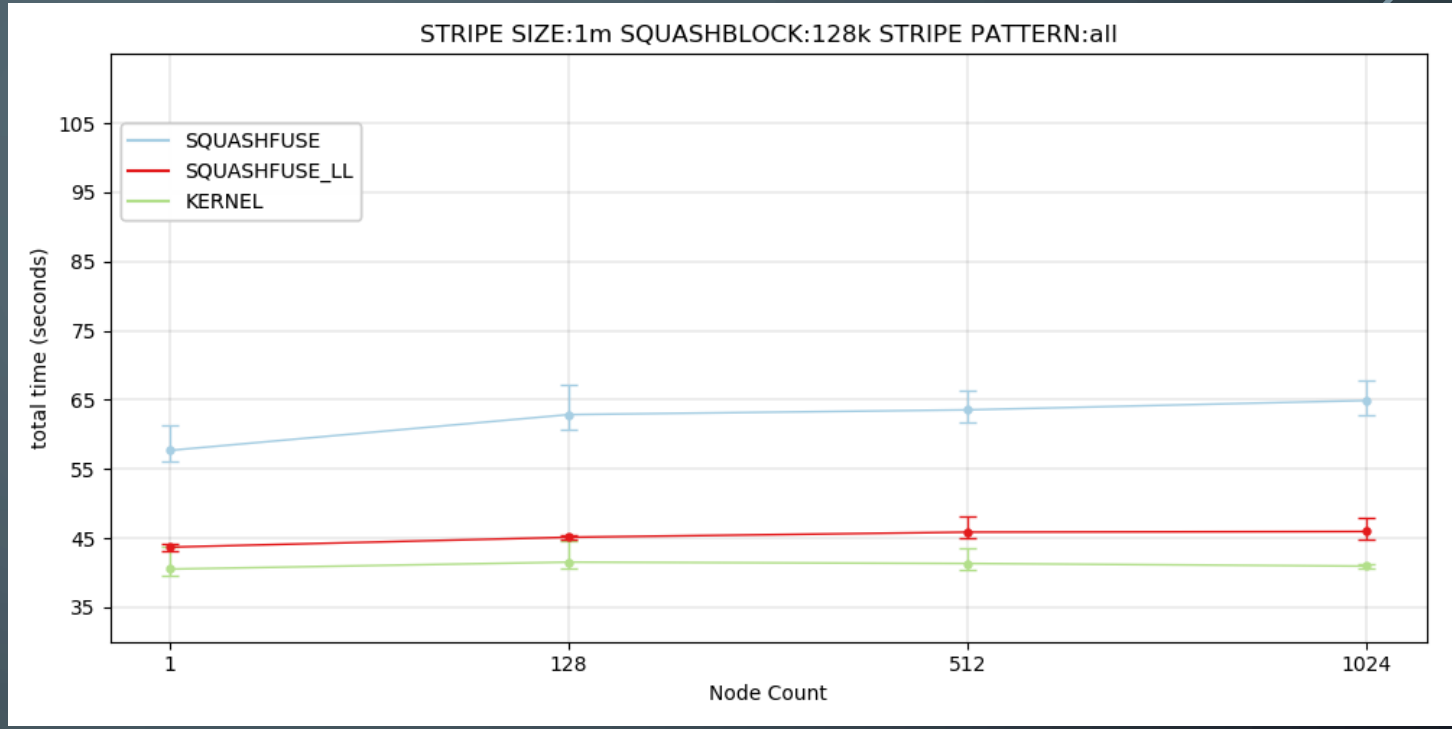


LUSTRE STRIPE SIZE

- Less impactful than the other factors
- The curve for the High-level FUSE API is slightly flatter with a smaller stripe size compared to larger
- Lustre stripe sizes larger than the default were not tested

FUSE OVERHEAD

- High-level API has more overhead
- Low-level API is very close in the reported benchmarks to a kernel mounted squashfs
- When optimized performance is maintained as scale increases
- FUSE overhead scales very well and is manageable



MISC TESTS

- Tests on NFS were short lived, at 64 nodes Pynamic took 10 minutes
- Recursive grep had some quirks
  - Using high-level FUSE API recursive grep was quite slow, but finished even at 256 nodes
  - Took 370 seconds at 128 nodes
  - Worked at 256 nodes, gave questionable output after that
  - The low-level API was fantastically faster, but at higher node counts hangs with no error
  - Took 100 seconds at 128 nodes
  - Failed at more than 128 nodes

CAVEATS

- FUSE has a default maximum request size of 128KiB
  - Could explain why larger block sizes appear to hurt performance, instead of help
- There were no attempts to optimize FUSE
- Sqaushfuse might be improveable
- Never figured out why the low-level API hangs on recursive grep
- There was no competition for resources through the experiment
  - This will never happen in the real world
- Did not use enough different block and stripe sizes

FUTURE WORK

- Testing should be done across a larger scale
- More block and stripe sizes should be tested
- FUSE can possibly be optimized
  - New versions support new features
  - Maximum request size can be increased
- Squashfuse has not been updated in 8 years, there may be improvements that have made their way to the kernel code
- Lustre monitoring

LUSTRE JOB STATS

- Collects Remote Procedure Calls made during job time
- Collects information regarding read sizes, and number of RPCs
- Our system does not appear to be bound by block or stripe size for reads
- Squashfs Kernel was not checked with Lustre stats
  - Can’t confirm kernel mounted squashfs is even making RPCs to Lustre

CONCLUSIONS

- Using default squashfs block size and Lustre stripe size is best and easiest
- Striping across OSTs becomes more important at scale
- FUSE overhead is rather small, and appears to stay that way with the low-level API
- Kernel mounted squashfs is fastest, but less secure due to root access

LA-UR-19-24149

QUESTION TIME