A decorative graphic on the left side of the slide, consisting of a network of thin, light blue lines and small circles, resembling a circuit board or a neural network, extending from the top to the bottom of the frame.

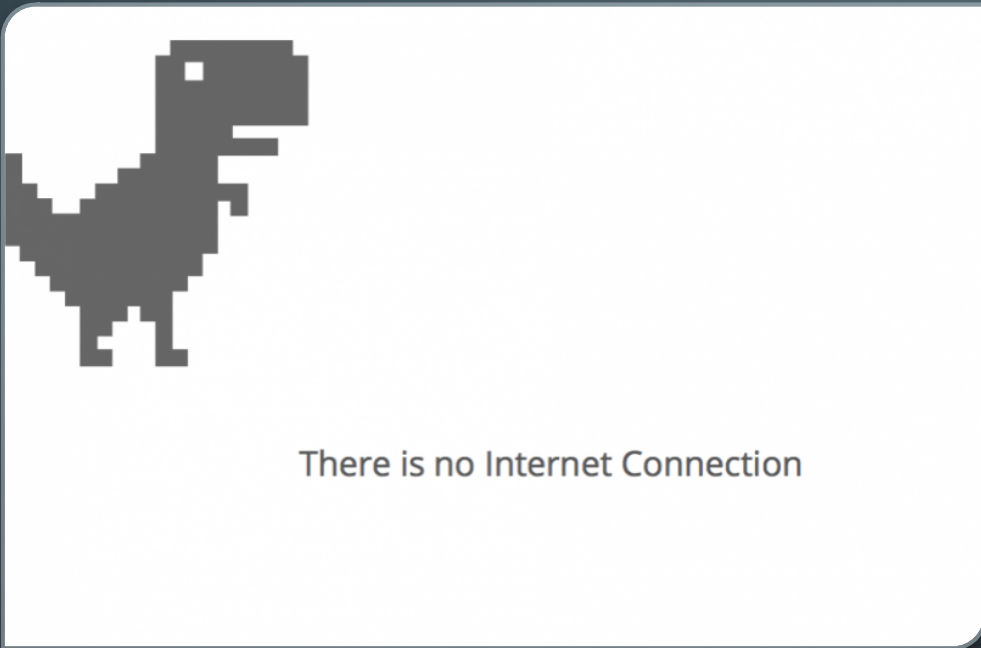
USERSPACE SQUASH FILESYSTEM FOR LAUNCHING LINUX CONTAINERS ON HPC SYSTEMS

BY RONALD SHANE GOFF

SOFTWARE IN HIGH PERFORMANCE COMPUTING

- Carefully administered and managed – very stable long term
- Only some versions of some things are installed
 - Might be missing an older or newer version you need
- 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 have numerous specific dependencies that are not supported
- Building software can be difficult
- 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

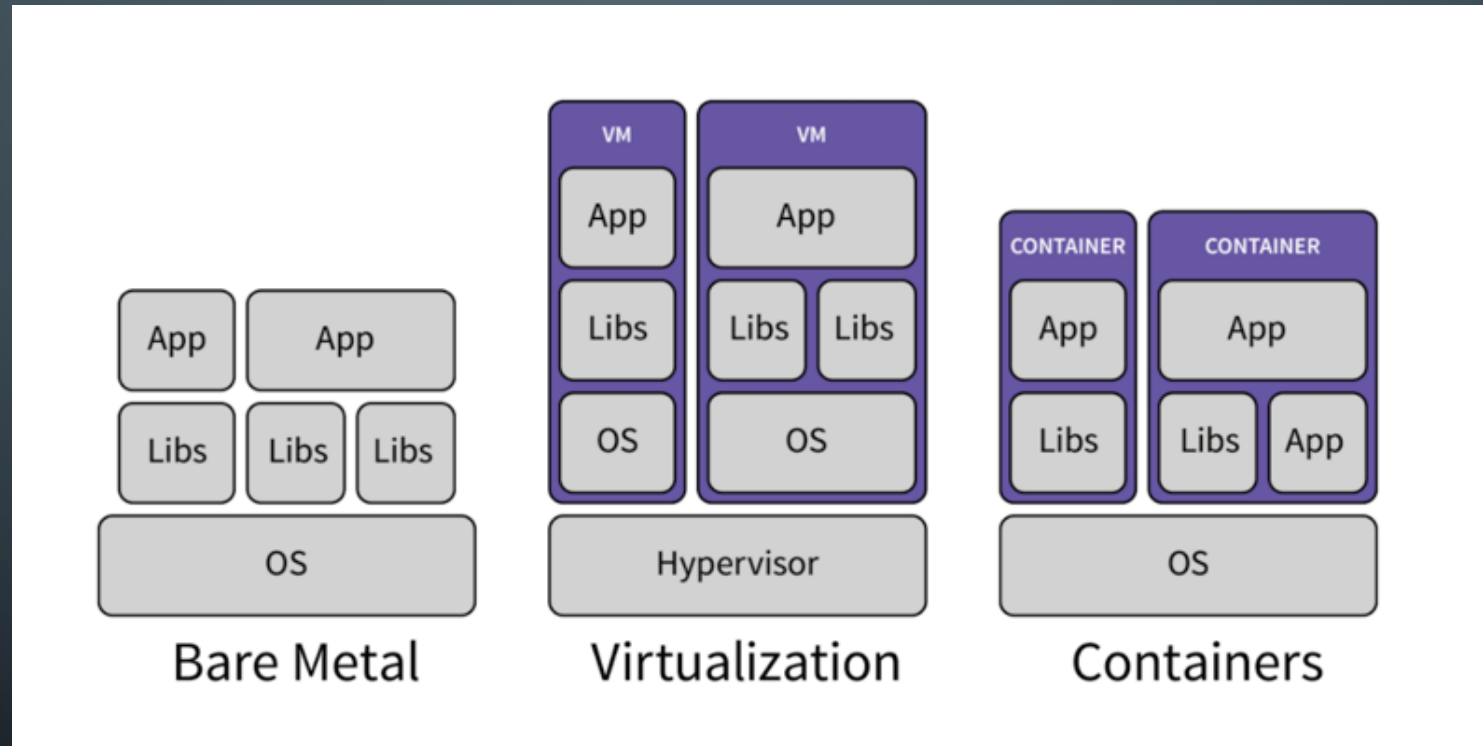
VIRTUAL MACHINES FOR ISOLATION

- Uses an entire guest operating system
- Separate user libraries and binaries
- 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
- Keeps host kernel safe from blunders

LINUX CONTAINERS

- Separates user binaries and libraries from the host
- Shares operating system kernel with the host
 - Can directly access special hardware
 - Smaller than virtual machines
 - More risk to the host
- Uses Linux name spaces to separate resources instead of a hypervisor
- Lower overhead not having to start an entire operating system

CONTAINERS ARE CLOSER TO BARE METAL



WHY CONTAINERS

- Fast
- Lightweight
- Unfettered internet access at build time
- Reproducible
- Portable



WE MUST ALSO ASK WHY NOT CONTAINERS



- User defined software stacks introduce new attack vectors
- Some container run time tools require root
- Containers can use more system memory than bare metal

UNPRIVILEGED WORKFLOW

- Containers by design do not need root to execute
- Some tools provide services to users that need root
 - Uses a root-level process to start user containers
 - Parent process also handles root-level things it needs to work
 - Concerning to most cluster admins, can expose root access to users
 - Extra services mean extra effort to support

RUNNING CONTAINERS IN HPC

- Some tools require root for running containers
 - Docker swarm
 - Singularity
 - Shifter
- Some do not
 - Charliecloud

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

OTHER WAYS TO DISTRIBUTE CONTAINERS

- You can run the container from a networked filesystem
 - No need to distribute containers manually
 - Slower execution time of containers, but less preparation time
 - Only copies necessary files instead of the whole container, using less 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
 - The reason Shifter requires root-level access
- Proven to be fast and efficient for containers in HPC
- Hosted on an existing network filesystem like Lustre

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

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

LUSTRE AND SQUASHFS

- Lustre doesn't deal well with small file transactions, due to having separate metadata
- 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


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
- High-level FUSE API
 - Wraps around low-level API, adds more overhead converting paths to inodes



HOW TO LUSTRE

Woah.



This is worthless!

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 DYNAMIC 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 application
- 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

BEST PRACTICES

PEOPLE

- 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. Even though containers won't do such a thing, someone will.

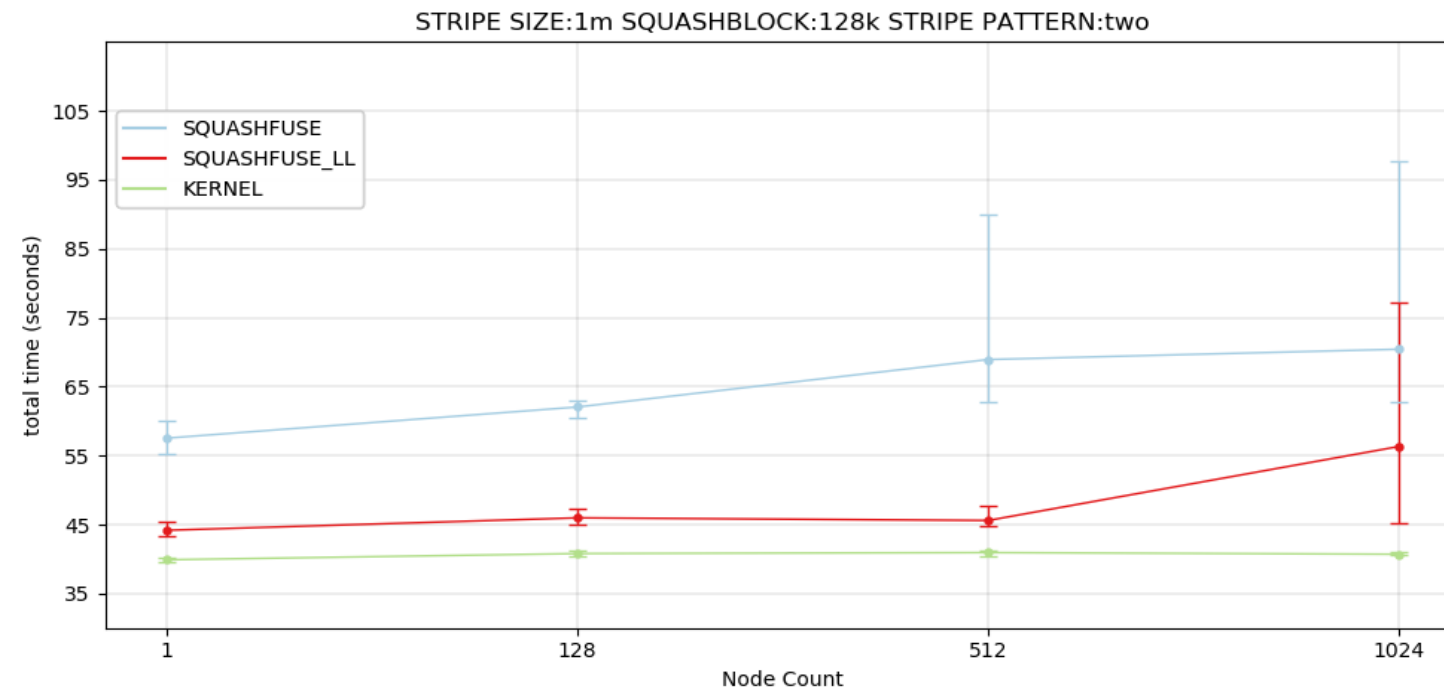
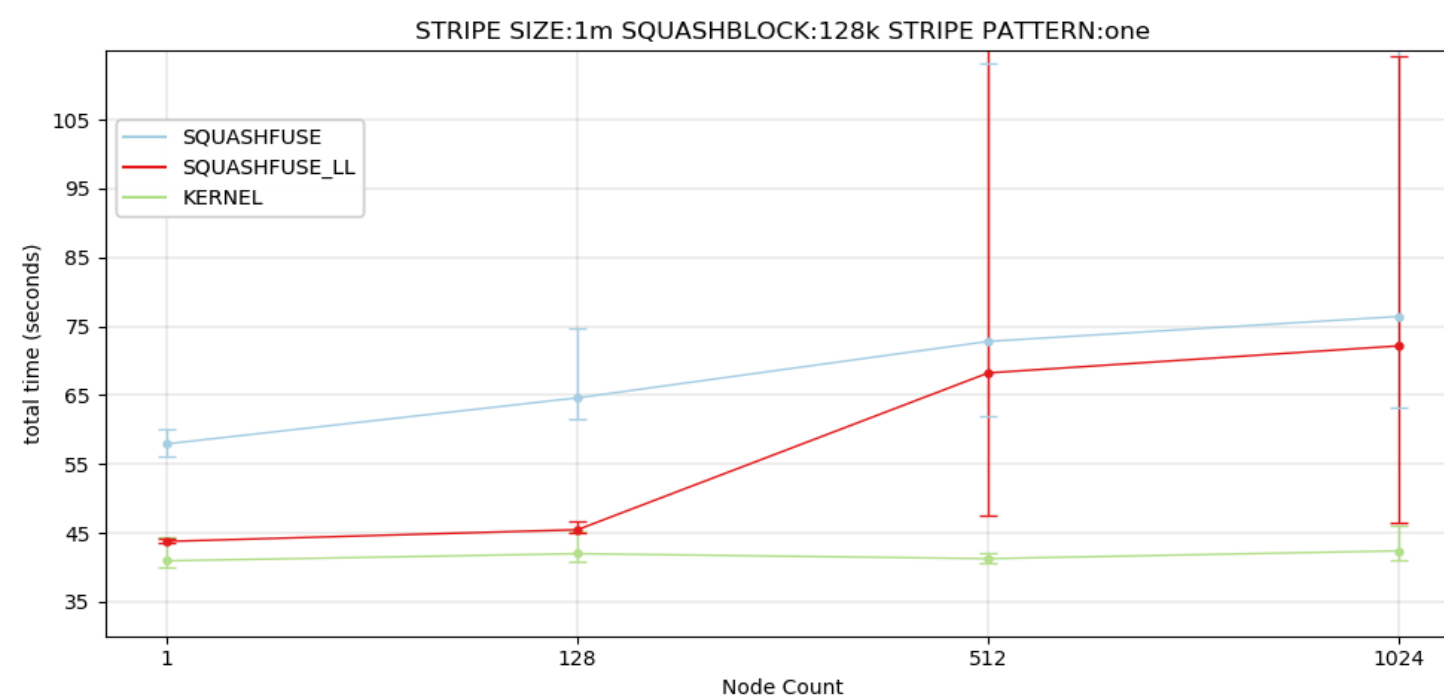
RESULTS

- 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

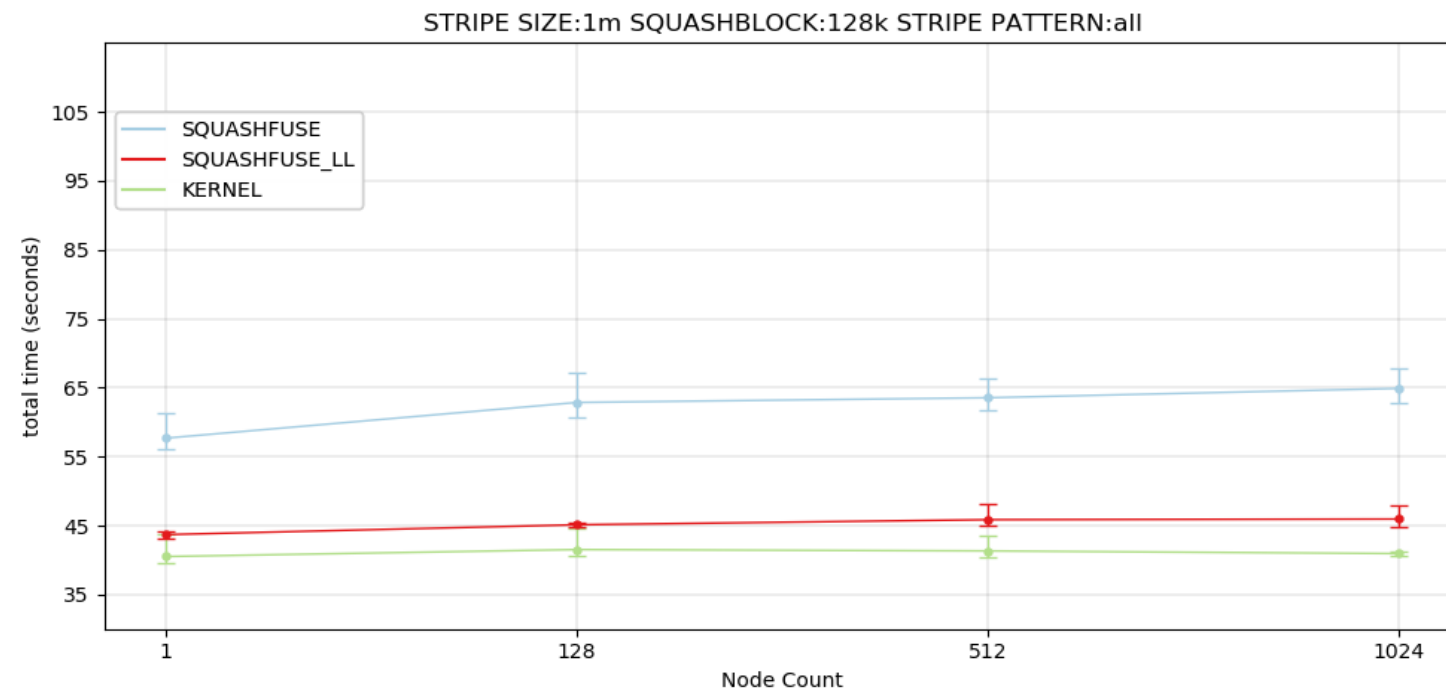
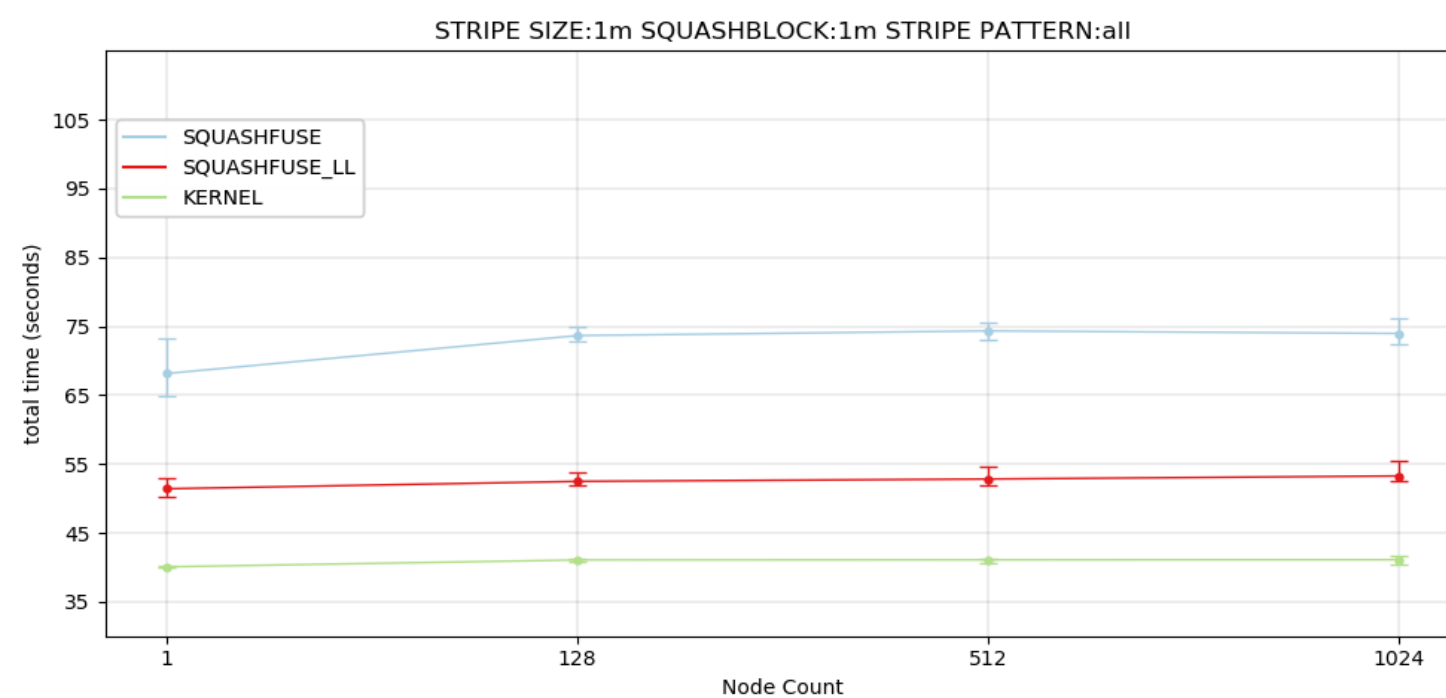
- Default squashfs block size, Lustre stripe size, and stripe count

- Increasing stripe pattern from one stripe, to two greatly improves performance as scale increases



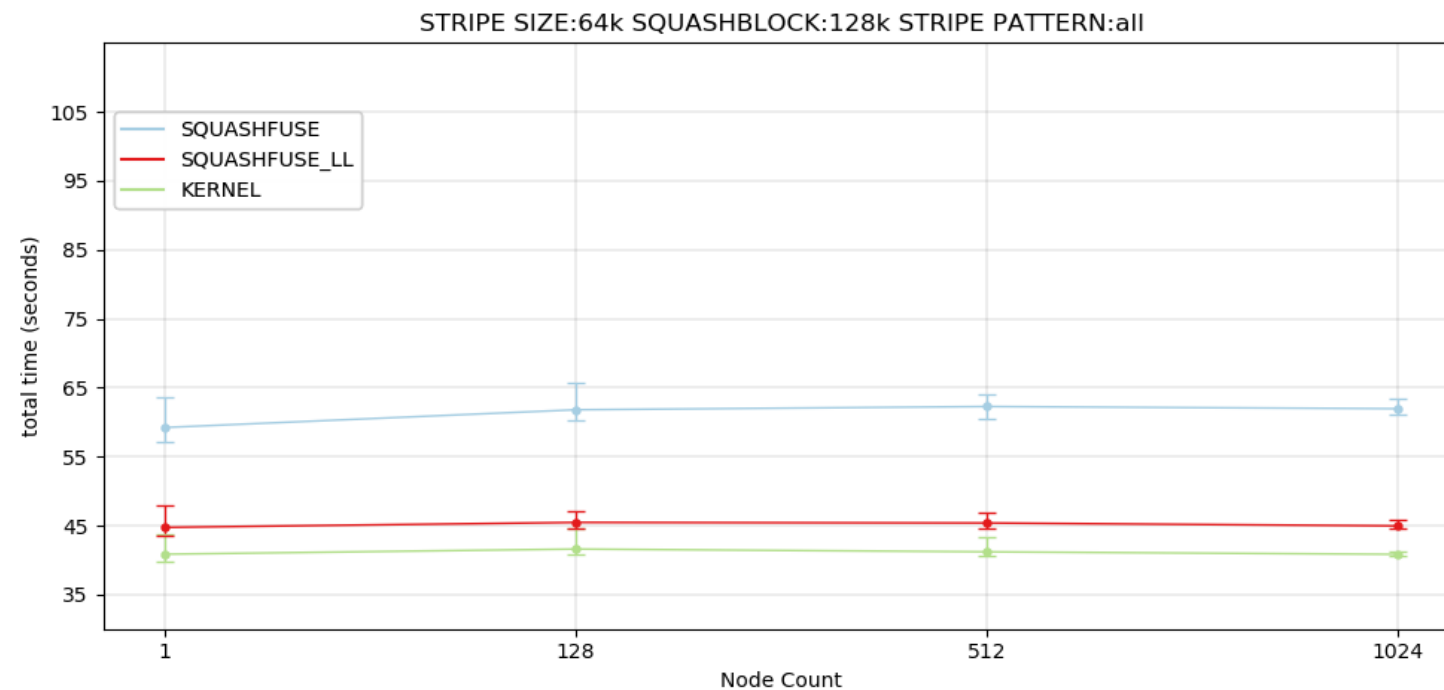
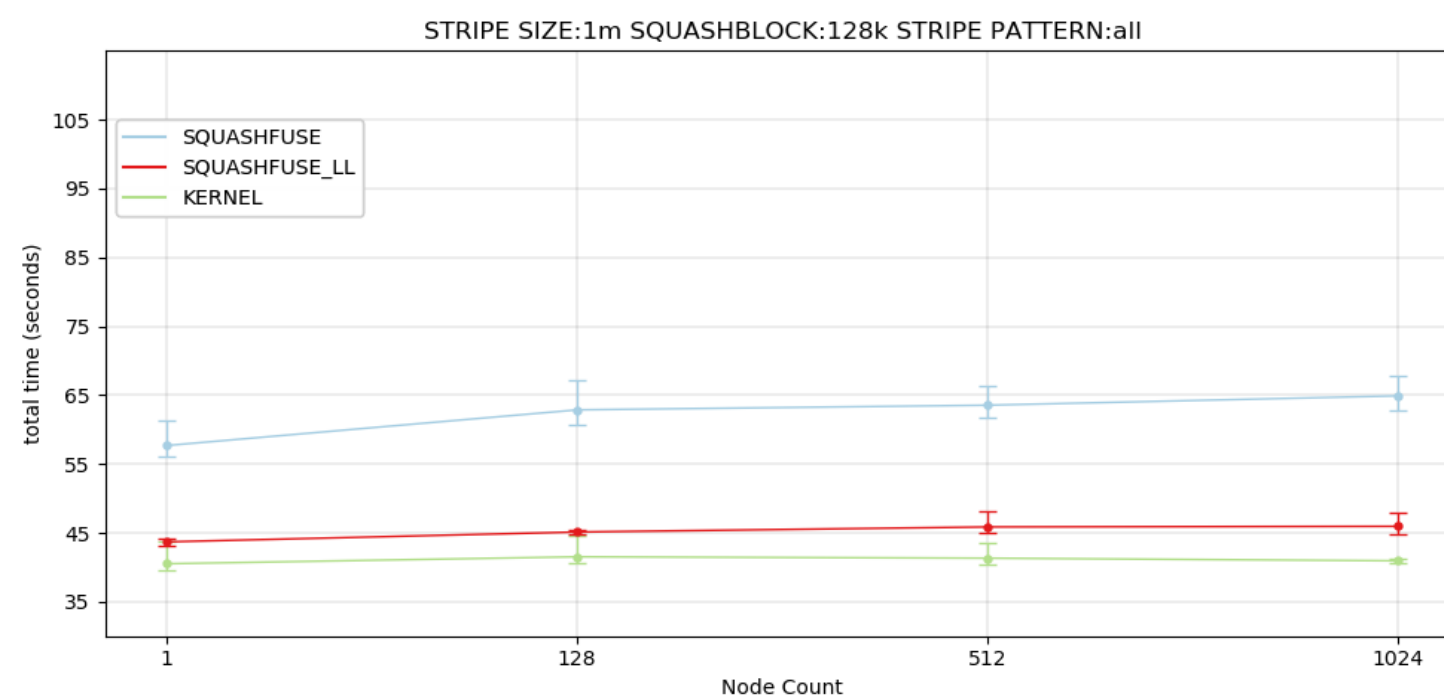
SQUASHFS BLOCK SIZE

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



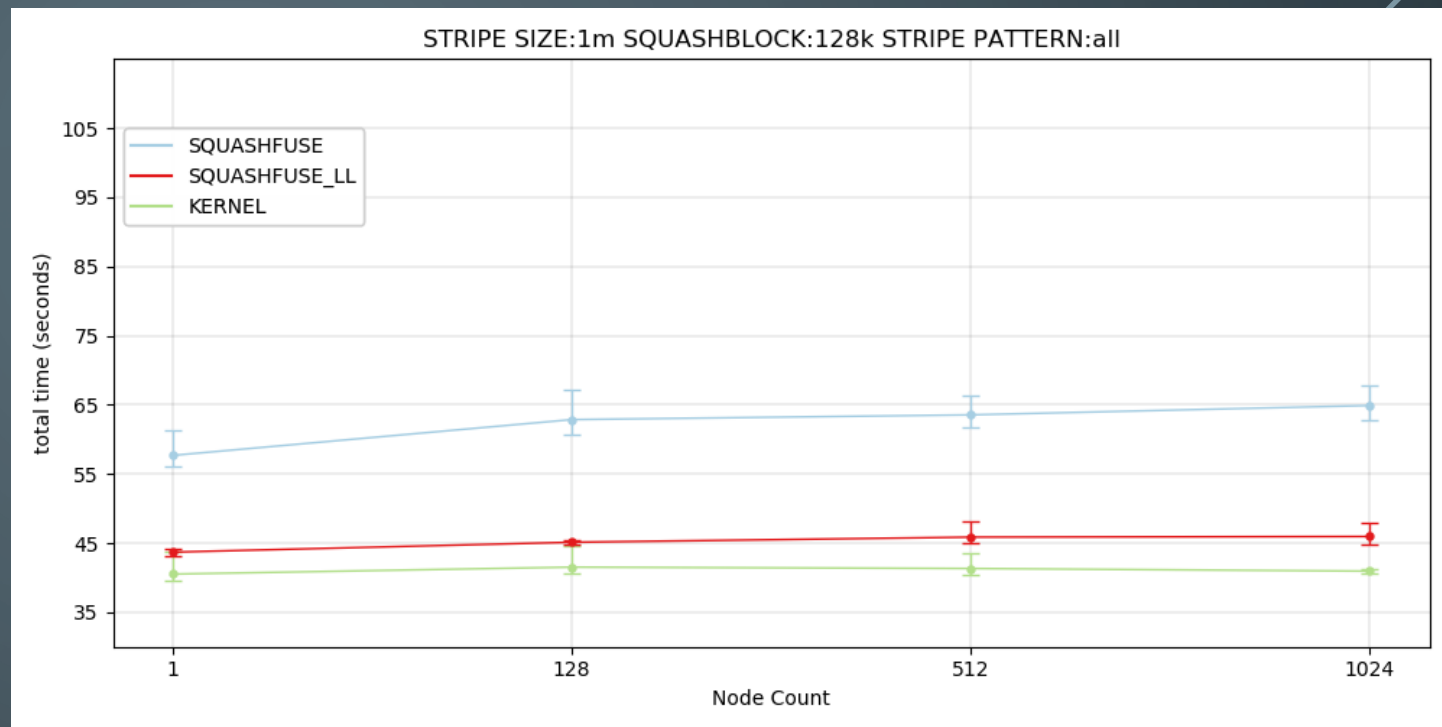
LUSTRE STRIPE SIZE

- 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
- 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

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

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

The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural networks. These elements consist of thin lines that branch out and terminate in small circles, creating a symmetrical, abstract pattern in each corner.

QUESTION TIME