**Thesis**

Ronald Shane Goff

The demand for Linux containers in high performance computing (HPC) has grown in recent years. In order to use containers in an HPC cluster, all compute nodes must have the container made available to them in some way. The squash file system (squashfs) has become quite popular for this task showing excellent performance. Squashfs is used in some tools as the default delivery methods for containers in HPC. However, this requires root privilege to accomplish in some way which can be undesirable in some data centers. Squashfuse is an existing tool created for using the squashfs without the need of root to mount the squashfs. Squashfuse provides disk operations through File Systems in Userspace (fuse) so any user can mount and make use of the squashfs. I believe squashfuse can bring adequate performance to containers without the security burden of root access. Furthermore, there are possible optmizations to the way squashfs is currently used by existing tools that can be explored. Squashfs and Lustre both have settings that can be changed to accomidate the disk operation patterns we see in containers.

In the HPC data center the software stack available to users falls into a few categories. There is a managed software stack provided by the cluster administrators which includes the most common and in demand tools such as message passing interface (MPI), common compilers, and more. Scientific software used in HPC can involve deep knowledge to properly set up and optimize for each computing cluster. It makes a great deal of sense for some of these tools to be managed centrally, ensuring top performance and stability for users. This means stable versions of common tools for users. However, users often have need of libraries and tools that are not centrally managed. There are even some users who require older versions of common tools that are no longer supported by the cluster administrator. If users want to use something that is not currently supported and maintained they have choices to make. They can either try to conform to the software stack provided, or find some way to work with the tools they want by building it themselves. Users can also request software be supported centrally, but often each user has specific needs and not all of them can be supported. If every tool a user wanted were to be centrally supported, the burden for the administration would be very high, to support a tool very few people may need. The path of least resistance is building the software you need that is not supported for yourself. Being able to build software you need beyond the supported tools is essential, as scientific software often makes use of a wide variety of libraries and tools. It is however not an ideal scenario for a few reasons. There are often more than one cluster available. If you built a large complex software stack on one machine, it is not useable on another without rebuilding all components. Some builds may also require internet access. As your software stack becomes more complicated so does rebuilding it. Containers are very valuable for improving this situation. Containers provide a build your own software stack to users. They are platform agnostic, and portable from one cluster to the next in many cases as everything needed is already in the container. This means reproducing your work on another machine is very simple. Containers are often built in scripts, so sharing your work is simple as sharing the container build process with others. As scientific software needs grow, containers will become more valuable for users. Being able to bring your own software to basically any machine quite easily becomes more valuable as your software stack becomes larger and more complex.

Isolation of your software stack has been performed by other methods as well. Machine virtualization has been very valuable to many users. You can create a virtual machine with your entire software stack already inside, and when the time comes you can spin up the virtual machine (VM) and away we go. This works great for some use cases. Even in the container world virtual machines have a place. For example, containers are often constructed on personal work stations. Users can create an isolated environment for themselves where they can have root access and everything required without risk to the host machine. In the cloud, virtual machine hypervisors have become very widely used for providing isolated remote machines accessed by users. This is great for giving users a space where they can have root, and do what they must. This is great for protecting other users from each other on shared hardware. If a user causes a crash, only their system is likely to be impacted. There is overhead involved with this process. Hypervisors require resources to provision and run virtual machines, as you are virtualizing an entire machine. Linux containers have less overhead as they share their kernel with the host. Only user libraries and tools are in the container. There is less isolation with containers than using a VM. This has perks, and risks. Virtual machines are completely isolated from the host by the hypervisor. In HPC there are often resources that are only available to the host that are very important. HPC systems have high performance networks, and filesystems in place. When you virtualize an entire machine, they do not have access to some of this vital hardware. As containers share the kernel with the host, using namespaces they still have access to things like the network and storage. Often in HPC efficiency is very important to users. Giving up performance to use VMs is unappealing when compared to containers. Containers provide other benefits as well. As they do not virtualize an entire machine, they are often smaller in disk size than VMs. With the ability to easily access valuable HPC hardware, smaller sizes, and improved efficiency over VMS containers are a clear winner for software isolation in HPC. As application run time doesn’t require root in most cases, there are very few reasons to use VMs over containers for HPC. There are several levels of virtualization available such as full virtualization, and paravirtualization which uses special in between layers to give the VM more access to local hardware. This still introduces overhead and problems, as the guest OS needs special code to use the paravirtualized devices. In a performance analasis of linux containers they “observed LXC is more suitable for HPC than KVM.” (Beserra et al., 2015) It would appear in all tests completed, linux containers beat virtual machines of all kinds in all tests. Even with paravirtualization providing more hardware access, containers proved to perform best. These tests were performed on a single node as well. As scale increases overhead from virtualization would likely increase as well.

There are a variety of tools already available to create and manage containers such as Docker, or BusyBox. Using Docker, it is quite easy to create containers that run HPC applications. However, Docker is not commonly used on HPC resources. This is because Docker requires root to create containers. This isn’t a problem, as building containers in an HPC environment would likely be more challenging anyway. HPC systems rarely have unfettered internet access. Access to the web is very useful for the automation of building a software stack. Any package or tool you need is available to you in an environment where you have internet access. This eliminates the need for downloading all the proper versions of your software in advance of build time. No more sending all of said software up to every machine you plan on using it on either. The build process for containers is much easier and more effective on a local work station, as opposed to the HPC data center. It is far simpler this way. Docker requiring root is also a deal breaker for most HPC data centers. Clusters are often very expensive, very carefully managed machines. Giving everyone root to build containers with Docker would be a hard sell for most cluster administrators. But, HPC resources are only needed at run time for containers, not for build time.

Containers are very popular in cloud and services use cases. There are a wide variety of tools specifically for running containers like Docker Swarm, Kubernetes, Redhat OpenShift, and more. There is no shortage of tools for container run time. These tools are however not designed with HPC in mind. These tools are more concerned with keeping services running, recovering containers from crashes, and scheduling of resources. These are great for running services, or hosting containers in a cloud. They are intended for the specific role of doing “something” as a service. Hardware as a service, API as a service, etc. But they do not serve HPC needs very well. The cloud is a rather large driving force for containers, as they have proved to be very valuable in that use. Some tools have also tried dipping their toes into running containers for HPC like Docker Swarm, but have been unpopular. Thankfully there are tools designed for HPC use cases available. Charliecloud is one of these tools. Charliecloud is developed at Los Alamos National Laboratory (LANL) designed to be lightweight, efficient, and secure for running containers on HPC resources. Other options include Shifter, which is developed at the National Energy Research Scientific Computing Center (NERSC). Shifter and Charliecloud serve the same purpose, they manage the run time process for launching containers in HPC. They also are very different. Shifter provides more features to users, such as access to container registries, and automation of container image creation.