Container workload and orchestration in high performance computing

by
Kees de Jong & Maxim Masterov

December 2, 2020

Abstract

In e.g. federated High Performance Computing (HPC) infrastructures it is a challenge to maintain predictable software environments. Container technology offers the portability needed to keep work environments across different infrastructures consistent. With container technology there is also the question of orchestration. How, where and when are these containers deployed in a (federated) cluster? Simple Linux Utility for Resource Management (SLURM) is a resource manager that is used to schedule HPC workloads and is used in about 60% of the HPC infrastructures in the TOP500. With SLURM, containers may be used in job scripts or with SLURM plugins to submit jobs, enabling the scheduling of HPC workloads with containers. In Cloud environments, Kubernetes is a popular container scheduler/orchestrator. This research investigated the pros and cons of several HPC oriented container solutions. Singularity was evaluated as the best all-round fit. Furthermore, SLURM and Kubernetes were evaluated in the context of HPC focused container schedulers, where it was clear that Kubernetes is making progress to better support HPC workloads. However, SLURM is deemed the best all-round choice.

Contents

1 Introduction **3**

1.1 Container technologies . . . 3

1.1.1 Singularity . . . 3

1.1.2 Docker . . . 3

1.1.3 udocker . . . 4

1.1.4 Charliecloud . . . 4

1.1.5 Podman . . . 4

1.1.6 Shifter . . . 5

1.1.7 Enroot . . . 5

1.2 Container orchestrators . . . 5

1.2.1 SLURM . . . 5

1.2.2 Kubernetes . . . 5

1.3 Summary . . . 6

2 Research question **6**

3 Method **6**

4 Results **7**

4.1 Container benchmarks . . . 7

4.2 Container technology in HPC . . . 8

4.2.1 Usability, features, and support for archived images . . . 9

4.2.2 SLURM and MPI support . . . 10

4.2.3 Support for recipes . . . 10

4.2.4 Cross-platform . . . 10

4.3 Container orchestrators in HPC . . . 10

4.3.1 Performance . . . 10

4.3.2 HPC workloads . . . 11

4.3.3 MPI support . . . 11

4.3.4 Resource management . . . 12

4.3.5 Infrastructure deployment methods . . . 12

4.3.6 Summary . . . 13

5 Conclusion **14**

1 Introduction

The use of containers in HPC environments is gaining more popularity. Container technology is used to provide a uniform environment for testing and deploying applications. Due to the bundling of all application dependencies into one portable container, these applications provide seamless and continues updates on any container-supporting host. This provides more agility because the same environment that is used by a developer will be used on any other system that supports that container format. However, adoption in HPC has been slow due to security concerns and the specific parallel Remote Direct Memory Access (RDMA) application use cases on InfiniBand networks.

1.1 Container technologies

Several HPC oriented container technologies have been developed and the pros and cons will be briefly summarized in this section. The conclusion are in part based on related work [1–3].

1.1.1 Singularity

One of the most popular HPC oriented container solutions is Singularity. An estimated of 25,000+ systems are running Singularity at e.g. SURF, TACC, San Diego Supercomputer Center, and Oak Ridge National Laboratory. Root privileges are not required to run and build containers. This is made possible by the use of user namespaces, which maps the UID/GID of the root user inside the container, to an unprivileged ID outside of the container. However, with user namespaces enabled, the attack surface is widened [4].

Singularity uses Singularity Image Format (SIF) which is a single-image format, i.e. it does not use layers, unlike Docker. Since the Open Containers Initiative (OCI) format supports multiple layers, they are often larger than the SIF format. SIFs are treated like a binary executable. And thus are easy to use in HPC batch scripts. However, Singularity also supports the conversion from the OCI format to SIF, which in effect allows using Docker images. Furthermore, the project enjoys active development from a broad community.

1.1.2 Docker

Docker is not an HPC oriented container solution. The reason for that is that it is focused on sites with only trusted users. Docker requires root privileges to build and run containers, as also noted in the Docker documentation; "only trusted users should be allowed to control your Docker daemon" [5]. HPC systems are in general systems where the users are not trusted with this privilege. The Docker daemon runs as root, which is considered a poor design choice in terms of security. Since Docker Engine v19.03 a new feature has been introduced named rootless-mode. However, this feature is labeled as experimental and is focused on Ubuntu based systems. This rootless-mode feature also lists several known limitations [6].

Therefore, allowing users to run Docker containers on a multi-user host with shared file systems and NFSv3 exports secured only via AUTHSYS is risky. Since users may mount these shared file systems as root and access this data. The only safe way to give users Docker like functionality on a shared host is with e.g. Singularity [7].

Saha et al. researched the performance of Docker and Singularity in HPC [2]. The researchers presented 1) a performance evaluation of Docker and Singularity on bare metal nodes in the Chameleon cloud 2) a mechanism by which Docker containers can be mapped with InfiniBand hardware with RDMA communication and 3) an analysis of mapping elements of parallel workloads to the containers for optimal resource management with container-ready orchestration tools. They evaluated Docker and Singularity as follows; Singularity is designed to use the underlying HPC runtime environment for executing Message Passing Interface (MPI) applications, whereas Docker is designed to isolate the runtime environment from the host. Also, Singularity focuses on coarse-grained resource allocation whereas Docker can take advantage of the fine-grained allocation of resources per rank.

The researcher's performance analysis showed that scientific workloads for both Docker and Singularity based containers can achieve near-native performance. Singularity is designed specifically for HPC workloads. However, Docker still has advantages over Singularity for use in Clouds as it provides overlay networking and an intuitive way to run MPI applications with one container per rank for fine-grained resources allocation. Both Docker and Singularity make it possible to directly use the underlying network fabric from the containers for coarse grained resource. However, unlike Singularity, a Docker container needs to have InfiniBand interconnect drivers installed and mapped inside the container to enable fast communication. Furthermore, for MPI applications, splitting ranks per container with restricted resources to each container can be employed by Docker. This option is not available in Singularity containers.

After this report was already finalized, Docker 20.10 was released [8] which solved many of the security concerns expressed in this subsection. These new features have not been taken into consideration because this research was already closed by the time Docker 20.10 was released.

1.1.3 udocker

In response to the security concerns of Docker, several more security hardened alternatives were developed. E.g. udocker was developed, which is a Docker feature subset clone that is designed to allow execution of Docker commands without increased user privileges. udocker does not require any type of privileges nor the deployment of services by system administrators. It can be downloaded and executed entirely by the end user. udocker achieved this enhanced security functionality by executing containers completely in user space. Because of that, administrative functionality inside of the container is severely limited [9].

1.1.4 Charliecloud

Charliecloud is designed to be as minimal and lightweight as possible and uses Linux user namespaces to run containers with no privileged operations or daemons and minimal configuration changes. This simple approach avoids most security risks while maintaining access to the performance and functionality already on offer. Charliecloud was not deemed stable enough for Red Hat Enterprise Linux (RHEL) due to the dependence on kernel namespaces in 2017 [10]. However, RHEL 8 is shipped with Podman (Red Hat's own container solution), which also makes use of kernel namespaces. It does not require root privileges to install the Charliecloud software or to run Charliecloud containers.

1.1.5 Podman

Podman was developed by Red Hat as a root-less container solution. It is designed without the overhead and security concerns of the full Docker daemon. Currently Podman is not entirely suitable for HPC use cases:

- Missing support for parallel filesystems (e.g. IBM Spectrum Scale).
- Rootless Podman was designed to use kernel user namespaces which is not compatible with most parallel filesystems.
- Not yet possible to set system wide policy defaults.

- Pulling and building Docker/OCI images requires manual subuids/subgids entries for each user.

Buildah offers a promising way to enable users to build container images as Docker/OCI images, all without root privileges.

1.1.6 Shifter

Shifter is mostly backed by the National Energy Research Scientific Computing Center (NERSC) and Cray. Documentation uses SLURM for job scheduling. However, instead of the OCI, Shifter uses their own format, which is reverse-compatible with the OCI format. Community support lacks for Shifter, other than NERSC and Cray there are not many other contributors, which indicates low engagement of the HPC community. This translates in low development activities, a pull request for better MPI integration, which was opened in April 2017, has since stalled.

1.1.7 Enroot

Enroot can be thought of as an enhanced unprivileged chroot. It uses the same underlying technologies as containers but removes much of the isolation they inherently provide while preserving filesystem separation. This approach is generally preferred in HPC environments or virtualized environments where portability and reproducibility is important, but extra isolation is not warranted.

Enroot is also similar to other tools like proot or fakeroot but instead relies on more recent features from the Linux kernel (i.e. user and mount namespaces), and provides facilities to import well known container image formats (e.g. Docker). Furthermore, it does not require a daemon or extra process. Several advanced features include; runfiles, scriptable configs, and in-memory containers [11]. Root privileges are required to build containers with enroot. However, Buildah from Red Hat may be used to build OCI containers as an unprivileged user, which can then be converted to the enroot format.

1.2 Container orchestrators

1.2.1 SLURM

SLURM provides the means to allocate exclusive and/or non-exclusive access to typically HPC compute resources for a duration of time. Therefore, SLURM provides a scheduling framework for starting, executing, accounting and monitoring compute jobs. These are typically parallel MPI jobs on a set of scheduled compute nodes, or parallel OpenMP jobs on a single scheduled compute node. SLURM also provides the intelligence to manage queues and thus congestion of the compute resources. SLURM is also topology-aware, which is the intelligence to schedule jobs on nodes close together in the HPC cluster in order to keep latency low. The HPC use case consists of parallel compute jobs with a defined, relative short runtime. SLURM is generally agnostic towards container technologies and can handle most, if not all.

1.2.2 Kubernetes

Kubernetes has risen to the top in the challenge to provide orchestration and management for containerized software components due to its rich ecosystem and scaling properties. Kubernetes provides portability, ease of administration, high availability, integrability, and monitoring capabilities for container orchestration. While HPC workload managers are focused on running distributed memory jobs and support high-throughput scenarios, Kubernetes is primarily built for orchestrating containerized microservice applications. Kubernetes has grown popular in e.g. Cloud-native workloads, high-throughput computing and data analytics workflows. These microservices require resilience, which Kubernetes may provide with load-balancing and redundancy features. In order to provide the maximum availability over a relative long lifespan, microservices are updated and maintained while in production [12], in contrast to HPC jobs which have a relative short and static lifespan.

1.3 Summary

In this section we have introduced the reader with the different container technologies and two orchestrators. It was pointed out that Docker is not ideal for HPC environments since it requires trusted users. udocker showed significant progress in terms of security by running containers fully in userspace, however, this limited its functionality. Charliecloud is a secure container solution with a small attack surface due to its lightweight nature. Charliecloud also does not require root privileges to build and run a container. Podman shows much promise due to its ability to also build and run containers without root privileges and without much overhead. However, at the time of writing this report, Podman lacks the HPC oriented features. Shifter lacks community support and its current development activities are too low to pursue it further. Singularity is a secure and very popular container solution which allows to build and run containers without root. Enroot has a different approach, the project mixes different isolation methods, while staying lightweight, with builtin GPU support.

The two container orchestrators discussed in this section can be summarized as follows. SLURM is focused mainly on scheduling a distributed parallel compute jobs with a defined end time. Where it is specialized in customizable efficient partitioning, queuing, accounting, monitoring and prioritizing jobs while being topology-aware. While Kubernetes is mainly focused on keeping microservices up and running without interruption. Where Kubernetes is specialized with redundancy features where for instance a new instance of a container is spawned when failures occur (high availability). Kubernetes also includes load-balancing features between containers to mitigate congestion and latency for the microservice. There are developments towards support for HPC workflows. However, Kubernetes still lacks the features to fully support these HPC workflows on an equivalent level as SLURM.

2 Research question

The following research questions are formulated and tested. How do these container technologies compare in terms of usability, security, features, and performance on single and multi node compute jobs? Furthermore, how to orchestrate/schedule compute jobs with containers? How do Kubernetes and SLURM compare with each other in terms of usability, scheduling features, and resource allocation?

3 Method

In order to answer the research question from section 2, we tested three linear solvers from PETSc library [13]: CG, BiCGSStab and AMG. The linear solvers were applied to a standard 3D Poisson problem with 7-point stencil and 500^3 Degrees of Freedom (DoF). The chosen linear solvers cover the most frequently used algebraic operations from the back-ends of most scientific codes. Among many, the most important operations are dot product, matrix-vector product, vector update, and matrix-matrix multiplication. By choosing these linear solvers, we intended to cover the majority of the possible use cases from fields like Computational Fluid Dynamics, Mechanical Engineering, Astrophysics, Machine Learning, and many others.

Every test was executed on 1, 2 and 4 nodes with 24 MPI tasks per node with hyperthreading switched off. To determine the deviation of the performance, every test was repeated four times. Table 1 shows versions of the libraries and compilers used during the tests in the bare metal case and inside the containers.

	Host machine	Container
OS	RHEL7.8	Ubuntu-18.04.4 LTS
GCC	7.3.0	7.5.0
OpenMPI	3.1.1	3.1.1
PETSc	3.11.2	3.11.4

Table 1: Versions of the compilers and libraries used on the bare metal and in the containers during the tests.

The performance metrics were visualized in box plots to demonstrate the performance stability compared to the bare metal performance. Furthermore, the usability, security and HPC features of the container technologies were visualized in spider graphs. We did not setup a Kubernetes cluster due to the sufficient related work available. The evaluation of Kubernetes and SLURM was a literature study of which the results were also visualized in spider graphs. These spider graphs of the container technologies and container orchestrators visualize the pros and cons of each solution.

4 Results

In this section the container benchmark results are evaluated. The container orchestrators are evaluated exclusively based on the related work.

4.1 Container benchmarks

Figure 1 shows the performance comparison of the benchmarks executed with different container solutions and the bare metal. The box plots demonstrate performance consistency between four repetitive executions of each benchmark.

On four nodes, the deviation of the performance between the bare metal and containers is rather negligible at the reported time scales. However, on one and two nodes, the containers demonstrate slightly better performance compared to the bare metal. One of the reasons for the performance deviation can be attributed to different versions of the GCC compiler and PETSc library used on bare metal and in containers. Another reason can be related to the better resource isolation in the containers and to the interference of the background processes on the bare metal system. With the exception of the AMG solver, the test performed in Charliecloud shows slightly higher elapsed time compared to other containers. This might be an indicator of the additional constraints imposed by Charliecloud on network communication.

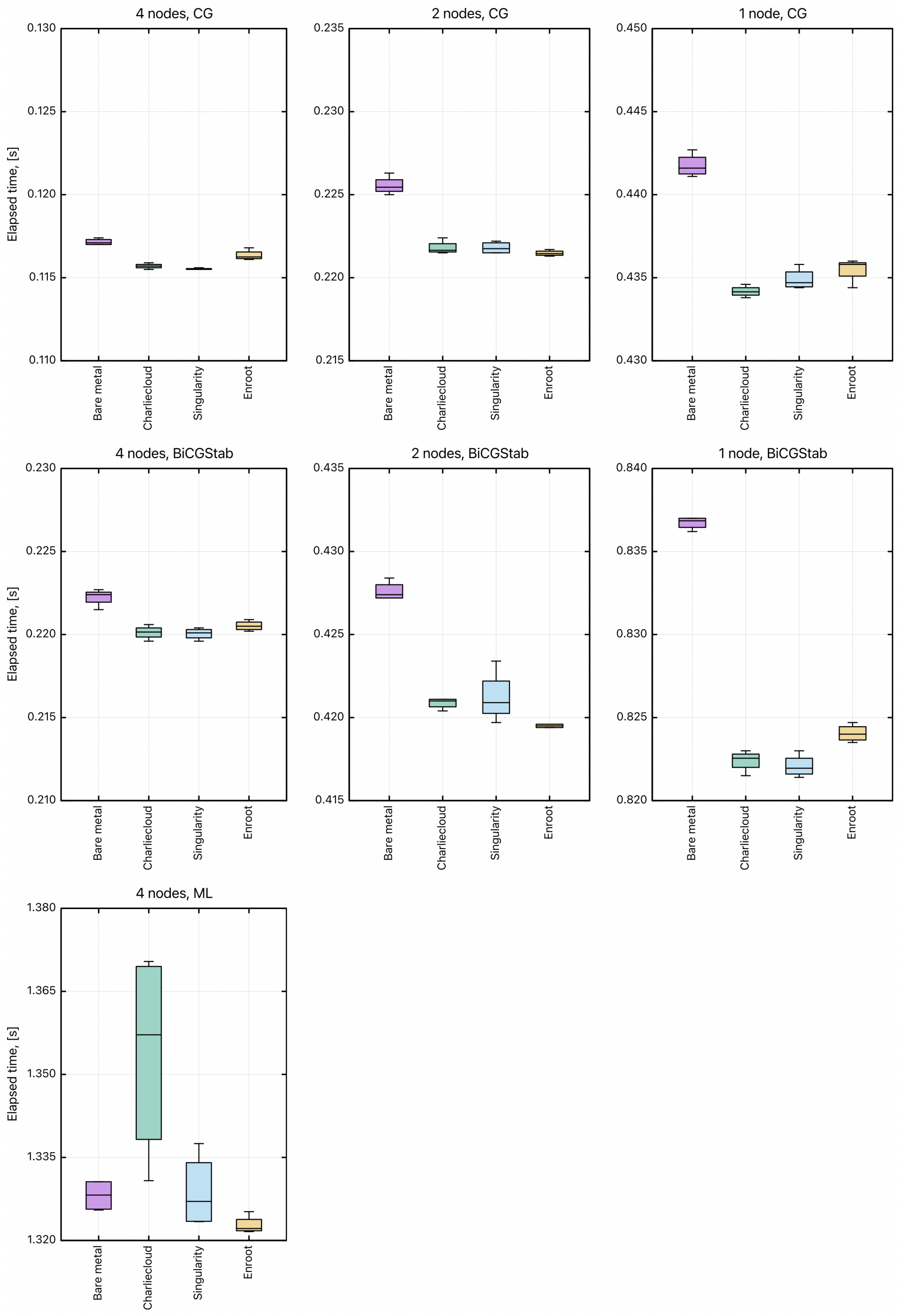

Figure 1: Comparison of the elapsed time of different linear solvers between different container technologies and bare metal. The benchmarks were executed on four (left), two (middle) and one (right) nodes with 24 cores per node.

4.2 Container technology in HPC

Figure 2 shows the strengths and weaknesses of the evaluated container technologies. The figure is based on an evaluation of nVidia [11] and represents the most important aspects of the container technologies on a scale from 0 to 5, where 0 is defined as bad and 5 as good. Based on our findings in section 1, we concluded the same results as nVidia for the following subjects: “Low overhead”, “Weak isolation”, “GPU support”, and “Security”. Furthermore, “Rootlessness” is fully based on our evaluation in section 1. The remaining evaluations are based on section 1 and our experiences with the container technologies on our HPC system.

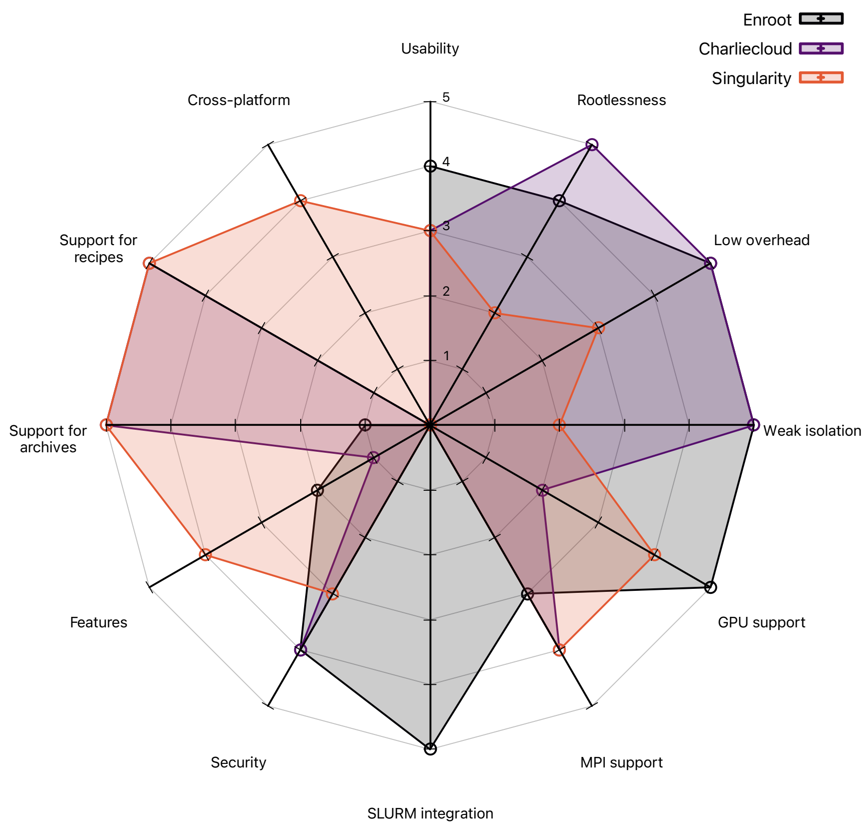

Figure 2: Strenghts and weaknesses of the tested container technologies.

4.2.1 Usability, features, and support for archived images

In terms of usability from the users’ perspective; Singularity has 78 manuals stored in `/usr/share/man/`, which reflects the amount of command line arguments available. This benefits Singularity in terms of features. However, it also makes it slightly harder to find the required information when a user does not know where to look. In comparison, Charliecloud is packaged with 20 manuals. Enroot went even further, by not including any manuals.

However, the online documentation of enroot was sufficient. Enroot’s features are more basic and thus getting to know the tool and finding out all the features was easier. We noticed one missing feature, i.e. the ability to create archived images (tarballs). This feature simplifies the utilization and execution of the containers that can be built off-site.

The usability in terms of the engineers’ perspective was not taken into account in this evaluation. However, we would like to note that Singularity and Charliecloud had the best support for RHEL systems. We had to build and maintain our own RPMs for enroot[1] [2].

[1]`https://fedorapeople.org/cgit/keesdejong/public_git/rpmbuild.git/tree/SPECS/enroot.spec`
[2]`https://fedorapeople.org/cgit/keesdejong/public_git/rpmbuild.git/tree/SPECS/pyxis.spec`

4.2.2 SLURM and MPI support

In terms of HPC integration, i.e. MPI and SLURM support, enroot performed better by providing a SLURM plugin. The enroot Pyxis plugin significantly simplifies the utilization of the container technology on an HPC system. This, for instance, allows users to rely on system-defined rules for core affinity and core binding. Furthermore, all tested containers allow to compile and execute MPI applications in the container. However, enroot does not provide their own solution to build containers. Buildah (from Red Hat) is used to build a container without privileges. The built container then has to be converted to the enroot format.

4.2.3 Support for recipes

Singularity and Charliecloud support the utilization of the recipe files to specify a container build. Charliecloud allows the use of the industry-standard “Dockerfile”, whereas Singularity can operate only with recipe files of its own format. Enroot can only operate with images from registries and can only use a limited functionality of so-called configuration files.

4.2.4 Cross-platform

Only Singularity can be considered as a cross-platform container solution (with some limitations and complexities on Mac machines). The two other container technologies can be run only on Linux-based systems.

4.3 Container orchestrators in HPC

In this section we will discuss the strengths and weakness of SLURM and Kubernetes in HPC.

4.3.1 Performance

Futral researched the performance differences of several batch schedulers, including SLURM and Kubernetes [14]. The researcher’s measurements were wall clock time, RAM usage, and CPU usage. These measurements captured the utilization of system resources for each of the schedulers. The batch jobs were composed of custom scripts, using the NASA Parallel Benchmark programs and computational fluid dynamics, which were executed using, 1, 2, and 4 servers to determine how well a scheduler scales with network growth. All hardware was similar and was co-located within the same data-center.

Kubernetes performed less than SLURM. For instance, Kubernetes needed to determine if Weave-Net was the appropriate network plugin for the cluster before starting a container. This overhead resulted in a slower time-to-spool. Furthermore, Kubernetes did not perform well because the worker pods had to be first provisioned before a controller pod could be provisioned via the batch job. Kubernetes also consumed the most RAM. Furthermore, the Kubernetes cluster also suffered from utilizing SSH for its communication protocol. This was due to the extra overhead of encryption in SSH. SLURM does not use encryption, SLURM only authenticates communication with a distributed symmetric key (MUNGE). In addition to SSH, the Kubernetes setup also relied on a virtual network and custom dynamic DNS solutions to determine worker node availability. The added layer of the virtual network and the DNS lookups significantly affected Kubernetes’ its performance. In [14], SLURM performed best and did not appear to require any optimizations in terms of additional configuration of the supporting OpenMPI libraries themselves.

While Kubernetes does provide some batch job facilities, ease of development, and process isolation; the research concluded that it overall did not perform as well as expected. In conclusion, the data that was collected suggests that most batch schedulers are uniquely tuned to improve performance of high-performance compute jobs. This advanced tuning was especially pronounced in SLURM, and less pronounced with Kubernetes.

4.3.2 HPC workloads

"Embarrassingly parallel", or "perfectly parallel" applications on a Kubernetes cluster can launch multiple containers in parallel, but the scale is "best effort" at launch time as "gang scheduling" is not possible; also it is not possible for the application to perform different functions on the "primary" container as the "secondaries", so sharding and setup must be performed either in advance or manually. This is not architecturally compatible with most existing applications [15]. This makes a Kubernetes cluster not suitable as a tightly-coupled parallel solver as it cannot guarantee scale neither at launch nor at runtime.

Accelerated parallel training for AI on Kubernetes is not supported for the same reasons as for parallel solvers; alternative training workflows must be used (assuming framework support) when scale is needed. However, AI workloads for accelerated real-time analytics is supported. Assuming Kubernetes plugins exist for accelerated hardware, it may support these stacks.

Many HPC applications have specific requirements relative to where they are executed within the system. Where each task (rank) of an application may need to communicate with specific neighboring tasks and so prefer to be placed topologically close to these neighbors to improve communication with these neighbors. This is called topology-aware scheduling. Other tasks within the application may be sensitive to the performance of the I/O subsystem and as such may prefer to be placed in areas of the system where I/O throughput or response times are more favorable [16]. As of writing this report, Kubernetes may only make this possible by manual scheduling containers on specific hosts in the cluster. In SLURM however, this is a builtin feature.

4.3.3 MPI support

SLURM also has builtin support for MPI such as Intel-MPI, MPICH2, MVAPICH2, OpenMPI, PMIx and UPC. MPI use depends upon the type of MPI being used. There are three fundamentally different modes of operation used by these various MPI implementation. 1) SLURM directly launches the tasks and performs initialization of communications through the PMI2 or PMIx APIs. 2) SLURM creates a resource allocation for the job and then mpirun launches tasks using SLURMs infrastructure. And 3) SLURM creates a resource allocation for the job and then mpirun launches tasks using some mechanism other than SLURM, such as SSH or RSH. These tasks are then initiated outside of SLURMs monitoring or control [17].

There are projects underway with the goal of integrating Kubernetes with MPI. One notable approach, kube-openmpi, uses Kubernetes to launch a cluster of containers capable of supporting the target application set. Once this Kubernetes namespace is created, it is possible to use kubectl to launch and mpiexec applications into the namespace and leverage the deployed OpenMPI environment. kube-openmpi only supports OpenMPI, as the name suggests.

Another framework, Kubeflow, also supports execution of MPI tasks atop Kubernetes. Kubeflow's focus is evidence that the driving force for MPI-Kubernetes integration will be large-scale Machine Learning. Kubeflow uses a secondary scheduler within Kubernetes, named kube-batch, to support the scheduling and uses OpenMPI and a companion SSH daemon for the launch of MPI-based jobs. Such approaches do not fully leverage the flexibility of the elastic Kubernetes infrastructure, or support the critical requirements of large-scale HPC environments.

In some respects, kube-openmpi is another example of the fixed use approach to the use of containers within HPC environments. For the most part there have been two primary approaches. Either launch containers into a conventional HPC environment using existing application launchers (e.g., Shifter, Singularity, etc.), or emulate a conventional data parallel HPC environment atop a container deployment orchestrator with kube-openmpi.

4.3.4 Resource management

Kubernetes 1.18 enables a new feature called Topology Manager [18]. This is a component of kubelet which also takes care of NUMA optimizations, which runs locally on the target host. Topology Manager provides host local logic, which enables; 1) only host local decisions and 2) lead to wasted resources when pods needs to be re-scheduled many times to find a host which fits. The Topology Manager provides following allocation policies: none, best-effort, restricted, and single-numa-node. These are kubelet flags. The actual policy which is applied to the pod depends on the kubelet flag but also on the QoS class of the pod itself. The QoS class of the pod depends on the resource setting of the pod description. If CPU and memory are requested in the same way within the limits and requests section then the QoS class is guaranteed. Other QoS classes are best-effort and burstable. The kubelet calls so called Hint Providers for each container of the pod and then aligns them with the selected policy, like checking if it works well with the single-numa-node policy. When set to restricted or single-numa-node it may terminate the pod when no placement is found so that the pod can be handled by external means (like rescheduling the pod). Some interesting development efforts and functionalities already in place for Kubernetes [19].

- CPU Pinning and Isolation
- Device Plugins (GPUs, Infiniband, FPGA, etc)
- NUMA Management
- SR-IOV Networking plugin
- Spark 2.3.0 with Kubernetes scheduler instead of Yarn
- Pods Priorities and Preemption
- Container Runtime
  - CRI-O - OCI stable runtime follow rootless
  - KataContainers - OCI VM Containers
  - Singularity - Syllabs seems to be working on support of Kubernetes
  - Gvisor - Google sandboxed containers
- Kubernetes cluster federation – for offloading on hybrid clouds
- Kubeflow to make machine learning simple, portable and scalable

4.3.5 Infrastructure deployment methods

The Kubernetes blog provides several suggestions for HPC deployments [20]. 1) Maintain separate infrastructures. For sites with sunk investments in HPC, this may be a preferred approach. Rather than disrupt existing environments, it may be easier to deploy new containerized applications on a separate cluster and leave the HPC environment untouched. The challenge is that this comes at the cost of siloed clusters, increasing infrastructure and management cost.

2) For sites running traditional HPC workloads, another approach is to use existing job submission mechanisms to launch jobs that in turn instantiate containers on one or more

target hosts. Sites using this approach can introduce containerized workloads with minimal disruption to their environment. Leading HPC workload managers such as Univa Grid Engine Container Edition and IBM Spectrum LSF are adding native support for containers. Shifter and Singularity are important open source tools supporting this type of deployment also. While this is a good solution for sites with simple requirements that want to stick with their HPC scheduler, they will not have access to native Kubernetes features, and this may constrain flexibility in managing long-running services where Kubernetes excels.

3) Use native job scheduling features in Kubernetes. Sites less invested in existing HPC applications can use existing scheduling facilities in Kubernetes for jobs that run to completion. While this is an option, it may be impractical for many HPC users. HPC applications are often either optimized towards massive throughput or large scale parallelism. In both cases startup and teardown latency have a discriminating impact. Latency that appear to be acceptable for containerized microservices today would render such applications unable to scale to the required levels.

4.3.6 Summary

In this section we evaluated Kubernetes' usability in an HPC environment as SLURM would be utilized. In terms of performance, Kubernetes is not performing optimally compared to SLURM. Furthermore, typical HPC characteristics such as support for NUMA management, GPUs, InfiniBand, and FPGU is developing. As well as MPI support, namely for large scale Machine Learning by the use of e.g. Kubeflow. However, these are not first class citizen functionalities of Kubernetes and thus are mostly based on 3rd party support. Some HPC specific characteristics still lack on Kubernetes, such as topology-aware scheduling.

Furthermore, by trying to leverage the best of SLURM into Kubernetes, the full potential of Kubernetes and SLURM are not utilized. Some Kubernetes and SLURM deployment strategies were suggested, either by having a separate cluster, a shared cluster, or merge the platforms. These deployment strategies depend on the specific use case of a cluster, where a balance needs to be found in terms of (economic) resource efficiency.

In figure 3 the strengths and weaknesses of SLURM and Kubernetes are visualized. This figure is based on an evaluation from nVidia [11], and we support these findings based on sections 1 and 4.3.

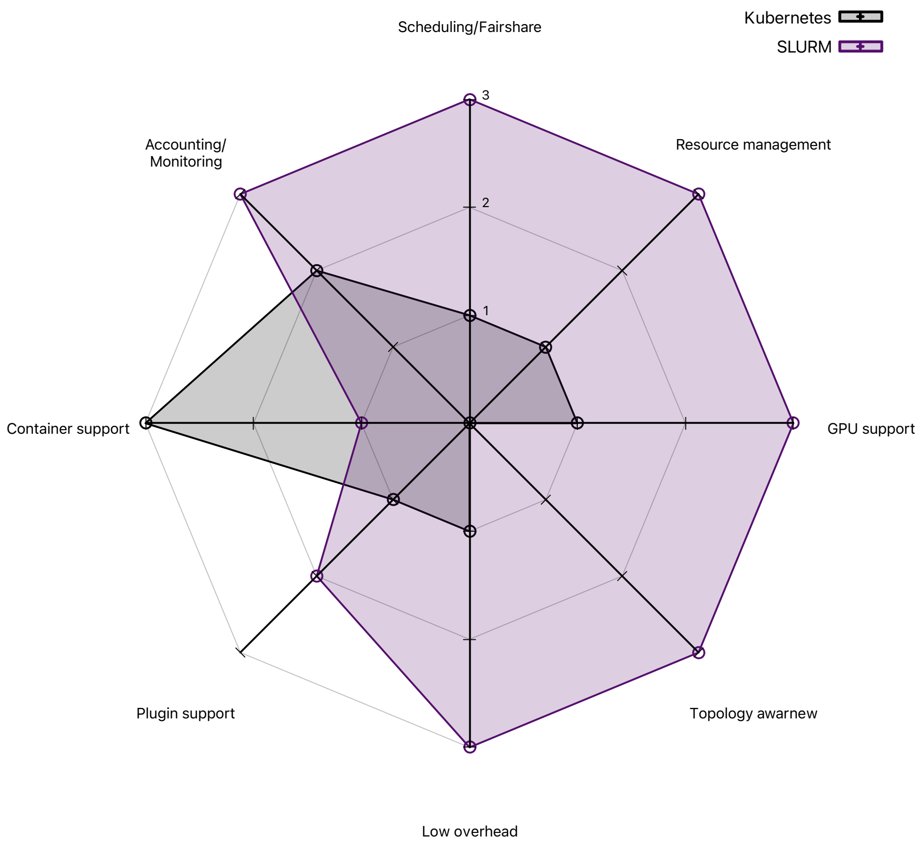

Figure 3: Strengths and weaknesses of SLURM and Kubernetes.

5 Conclusion

In this research report we evaluated container technologies and orchestrators in the context of HPC. Container usage in HPC has some unique requirements. Such as interconnect and GPU support, an unprivileged runtime for security, and for collaborating within the HPC community, an OCI standardized format. Furthermore, the preservation of cgroups (as is used in SLURM), ease of use with MPI workloads and the ability to modify containers for customization. The latter requires a setup like e.g. the fakeroot option in Singularity, or by using Buildah which allows unprivileged container builds.

From the container technologies we tested and evaluated, none really disappointed. In terms of performance there was no clear winner. All container technologies performed equally to bare metal performance. However, Singularity and enroot have the best overall HPC support (Figure 2). Where enroot especially excels for (nVidia) GPU workloads. Charliecloud looks very promising as well.

Traditional HPC and Kubernetes co-existence is possible. However, there should be a valid use case for such a more complex and expensive setup [7]. While Kubernetes excels at orchestrating containers, containerized HPC applications can be tricky to deploy on Kubernetes [20]. Kubernetes is not designed for HPC, hence it will never be as optimized as SLURM. However, in the effort to enable support of Big Data and Artificial Intelligence, Kubernetes is being enhanced with functionalities that will eventually interest the HPC community. Furthermore, the trend of convergence between HPC and Big Data may further motivate the usage of Kubernetes in HPC in the future.

References

[1] HPC workloads in containers: Comparison of container run-times. Justin W. Flory's blog. [Online]. Available: https://blog.justinwflory.com/2019/08/hpc-workloads-containers/

[2] P. Saha, A. Beltre, P. Uminski, and M. Govindaraju, "Evaluation of docker containers for scientific workloads in the cloud," in *Proceedings of the Practice and Experience on Advanced Research Computing*, 2018, pp. 1–8.

[3] The State of HPC Containers. StackHPC Ltd. [Online]. Available: https://www.stackhpc.com/the-state-of-hpc-containers.html

[4] CVE-2020-14386. Red Hat, Inc. [Online]. Available: https://access.redhat.com/security/cve/cve-2020-14386/

[5] Docker security. Docker Inc. [Online]. Available: https://docs.docker.com/engine/security/security/

[6] Run the Docker daemon as a non-root user (Rootless mode). Docker Inc. [Online]. Available: https://docs.docker.com/engine/security/rootless/#known-limitations

[7] Cloudy Topics from the Hutch. Fred Hutch. [Online]. Available: https://www.fredhutch.org/en/events/partly-cloudy/_jcr_content/root/responsivegrid_1/panelcontainer_206790373/contents/downloadpdf_1916703528/file

[8] New features in Docker 20.10. NTT Open Source / Medium. [Online]. Available: https://medium.com/nttlabs/docker-20-10-59cc4bd59d37

[9] udocker setup. The university of Utah. [Online]. Available: https://www.chpc.utah.edu/documentation/software/udocker.php

[10] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PloS one*, vol. 12, no. 5, 2017.

[11] SLURM: Seamless Integration With Unprivileged Containers. nVidia. [Online]. Available: https://slurm.schedmd.com/SLUG19/NVIDIA_Containers.pdf

[12] Kubernetes, Containers and HPC. HPCwire. [Online]. Available: https://www.hpcwire.com/2019/09/19/kubernetes-containers-and-hpc/

[13] S. Abhyankar, G. Betrie, D. Maldonado, L. McInnes, B. Smith, and H. Zhang, "Petsc dmnetwork: A library for scalable network pde-based multiphysics simulations," *ACM Transactions on Mathematical Software*, vol. 46, no. 1, 2020.

[14] J. S. Futral, "A method of evaluation of high-performance computing batch schedulers," *University of North Florida*, 2019.

[15] HPC on Kubernetes: A practical and comprehensive approach. Nimbix. [Online]. Available: https://www.nimbix.net/kubernetes-white-paper

[16] Kubernetes, HPC and MPI. StackHPC Ltd. [Online]. Available: https://www.stackhpc.com/k8s-mpi.html

[17] MPI and UPC Users Guide. SchedMD. [Online]. Available: https://slurm.schedmd.com/mpi_guide.html

[18] Kubernetes Topology Manager vs. HPC Workload Manager. Grid Engine. [Online]. Available: https://gridengine.eu/

[19] KubeCon 2018. KubeCon. [Online]. Available: https://events.linuxfoundation.org/events/kubecon-cloudnativecon-europe-2018/program/schedule/

[20] Kubernetes Meets High-Performance Computing. The Kubernetes Authors. [Online]. Available: https://kubernetes.io/blog/2017/08/kubernetes-meets-high-performance/