Containerization on Petascale HPC Clusters

Amit Ruhela, Matt Vaughn, Stephen Lien Harrell, Gregory J. Zynda, John Fonner, Richard Todd Evans, Tommy Minyard
Texas Advanced Computing Center
Austin, Texas
E-mail : {aruhela, vaughn, sharrell, gzynda, jfonner, rtevans, minyard}@tacc.utexas.edu

***Abstract*—Containerization technologies provide a mechanism to encapsulate applications and many of their dependencies, facilitating software portability and reproducibility on HPC systems. However, in order to access many of the architectural features that enable HPC system performance, compatibility between certain components of the container and host are required, resulting in a trade-off between portability and performance. In this work, we discuss our early experiences running three state-of-the-art containerization technologies on the petascale Frontera system. We present how we build the containers to ensure performance and security and their performance at scale. We ran microbenchmarks at a scale of 4,096 nodes and demonstrate the near-native performance and minimal memory overheads by the containerized environments at 70,000 processes on 1,296 nodes with a scientific application MILC - a quantum chromodynamics code.**

***Index Terms*—Petascale, HPC, Containerization, Cloud Computing, Singularity, Charliecloud, Podman**

I. Introduction

Containerization is a powerful tool for scientific software development and portability across systems. It considerably reduces the time to build, test, and deploy applications by encapsulating code and dependencies together, allowing them to run on diverse platforms with minimal additional effort. HPC infrastructures provide tremendous computing capabilities along with optimized message communication actualized through advanced features like eager communication, shared memory, and Remote Direct Memory Access, etc., making them ideal for intensive scientific computation but challenging for software portability. Containers provide a promising way to hide system-level complexities, allowing researchers to focus on productive studies that include COVID-19 research, climate modeling, agriculture, healthcare, smart cities, e-commerce, deep learning, etc.

Several studies in the past have focused on the performance characterization of containerized workloads [1]–[6]. These studies, conducted at small problem sizes, indicate near-native performance by container-based solutions. However, none of the prior studies have comprehensively shown the performance, usability, and portability of state-of-the-art container approaches at medium and large scale. This motivated us to study the following two problems: **(1) Does the performance of container-based solutions on HPC clusters match bare-metal runs at varying problem scales? (2) What are the challenges and possible directions to exploit the state-of-the-art container techniques at massive scale?**

A. Contributions

To best of our knowledge, this is the first study investigating the performance of containers at HPC petascale. In summary, the main contributions of this paper are:

1) We present the challenges and possible approaches to build HPC clouds with container-based approaches.
2) We present the changes required to configure containerization approaches on HPC infrastructures.
3) We present the overhead of state-of-the-art containers at small, medium, and large workloads.
4) We establish the usability and portability of three user-defined containerization stacks (Singularity, Charliecloud, Podman) at various problem scales.
5) We compared the performance of state-of-the-art containers at a scale of 4,096 HPC nodes with MPI microbenchmarks.
6) We compared the performance of native and container environments on an HPC scientific application with 70,000 processes at 1,296 nodes.

II. Methodology

Out of several container runtimes, we evaluate Charliecloud, Singularity, and Podman in this work. The goal is not to investigate comprehensively but to manifest the simplicity, usefulness, and performance of a few popular container types at petascale clusters.

Containerization on HPC clusters is challenging mainly due to access privileges and security requirements. Further, batch processing of jobs along with container overheads adds unique challenges to their usability. Portability of containers is restricted by ABI compatibility between the container and host hardware driver libraries along with instruction compatibility with host architecture (high speed interconnect drivers, GPU drivers, processor ISAs, processor specific compiler optimizations). For actable non-optimal performance, the container need not utilize specialized drivers and hardware capabilities and only ISA portability is required.

Singularity is a container platform specifically crafted for HPC systems. Similar to other user space container systems, Singularity bind mounts a container image and changes the apparent root (chroot) to the container. Singularity goes a step further to support the HPC ecosystem by mounting native devices (e.g., GPU, network, IB) and configured filesystem paths while also preserving Linux namespaces and user mapping inside the container. Singularity does not run a daemon

service, but must be installed by the root user for privilege escalation. After building images from their own development systems, or on HPC if fakeroot is configured, users can pull images built with Singularity or Docker, and safely run them on shared HPC resources. While images can be stored in the cloud, they exist as single files on a filesystem, allowing them to be shared and managed like all other files.

Charliecloud, a user defined software stack (UDSS), exploits user and mount namespaces of Linux to run containers without needing privileged operations and/or daemons. Any packaging software capable of producing a standard Linux filesystem can build container images that can be hosted on private or public repositories (Dockerhub, Gitlab, NVIDIA NGC, etc.). Charliecloud is a 800 lines of open source code that demands minimal system control (sysctl) commands [7] to configure on computing facilities, which elude most security risks.

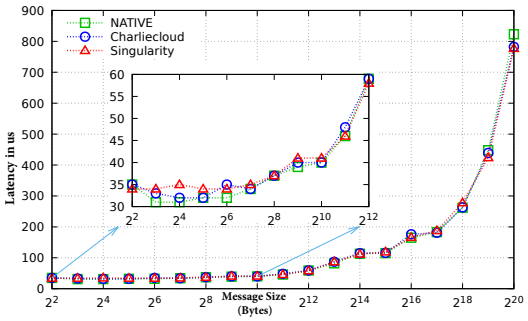
Podman is a new native container runtime. It builds and runs OCI-standard containers, but adds several attractive capabilities. Notably, it can run either individual containers or Kubernetes-style pods (orchestrated sets of containers) and it does so more safely and securely than Docker. As opposed to Docker's client/server approach (which requires privileged access), Podman uses a traditional fork/exec model. By leveraging user namespaces, root-level access is not required to run containers and additional isolation is enforced via UID separation. Podman is an attractive emerging technology since its CLI and user experience is nearly identical to Docker, which could make use of containers on HPC more accessible to end users. However, full use of Podman's rootless capabilities requires advanced kernel features such as version 2 cgroups and user-space FUSE, and is not yet compatible with network filesystems, which limited the extent to which we were able to evaluate it on a production HPC system.

III. Performance Evaluation

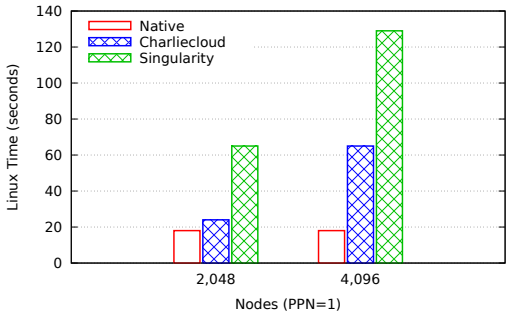
Each experiment was conducted 10 times on the Frontera supercomputer at the Texas Advanced Computing Center. Two benchmarks were used, first the Intel MPI Benchmarks [8] and second the SPP-2017 MILC Benchmark. [9].

Figure 1a shows the performance of "MPI_Allreduce" collective algorithm at 4K nodes. We observe that all three runtimes show nearly same latency at small, medium, and large message sizes. Figure 1b shows the total time required of running MPI_Allreduce benchmark, which includes the instantiating time of containers. Correlating the two figures exposes considerable overheads of instantiating container instances, but once the containers are available, their runtime performance stays on par with native runs.

Figure 2a and Figure 2b shows the performance in terms of CG_Time (time to solve conjugate gradient) and total time of native, Charliecloud, and Singularity runtimes with MILC application at 69,984 processes on 1,296 nodes. We observe that all three runtimes show similar time and incur negligible overheads. We also measured the memory consumption by MILC application in Figure 2c and observed that all three runtimes consume similar memory at various problem sizes.



(a) MPI_Allreduce



(b) Linux Time

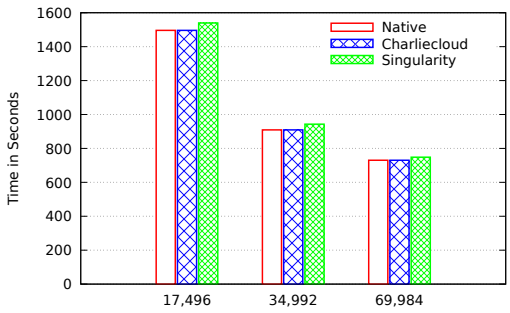
Fig. 1. Performance of Singularity and Charliecloud containers against native runs at 4,096 nodes

To explore the feasibility of using native containers in an HPC context, we ran the exact Docker containers from our large-scale studies on a test cluster configured to resemble Stampede2 (but with Linux kernel 5.8.1). All workloads ran correctly, albeit with minor (5-10%) performance degradation. We hypothesize the additional overhead is due to podman's use of fuse-overlayfs and its additional inter-process isolation, which may be resolvable with additional resource tuning. This experiment leaves us optimistic about future use of native containers on HPC.

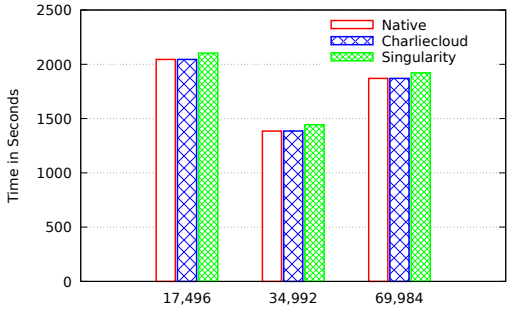
Our experiments with microbenchmarks and applications indicate that container solutions are an optimal choice for long-running applications. However, short lived applications are benefited from the containers when their build process is complex or time-consuming, and if computing platforms lack required functionalities to run the applications.

IV. Conclusion

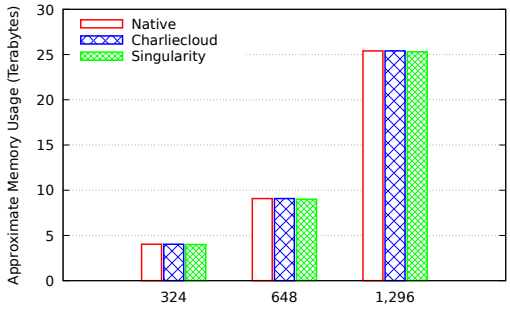
Recent technological advancements in containerization runtimes have commenced a new trend of HPC software development, which effectively reduces the build and deployment issues caused by complex software dependencies. In this work, we present the challenges of leveraging containerization within HPC systems and showcased the feasibility of three state-of-the-art container technologies. We explore the performance, usability, and portability of container workflows through experiments conducted at a petascale HPC cluster across tens of thousands of processes. We conclude that developers, testers, and end-users can leverage containerization on HPC systems



(a) CG_TIME



(b) TOTAL_TIME



(c) Memory Consumption

Fig. 2. Performance of Singularity, and Charliecloud containers against default run with MILC application at up to 70K processes

in a performant way, at large scale, to reduce software development and maintenance efforts. The cost of performance at scale is to build support for high-speed interconnects, such as InfiniBand, into the containers. This support does not, however, exclude their use in environments that only have more generic communications support such as TCP/IP or shared memory.

V. Acknowledgment

This work is supported by UT Austin-Portugal Program, a collaboration between the Portuguese Foundation of Science and Technology and the University of Texas at Austin, award UTA18-001217.

References

[1] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, "A tale of two systems: Using containers to deploy hpc applications on supercomputers and clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 74–81.

[2] C. Ruiz, E. Jeanvoine, and L. Nussbaum, "Performance evaluation of containers for hpc," in *Euro-Par 2015: Parallel Processing Workshops*, S. Hunold, A. Costan, D. Giménez, A. Iosup, L. Ricci, M. E. Gómez Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, and M. Alexander, Eds. Cham: Springer International Publishing, 2015, pp. 813–824.

[3] C. Arango, R. Dernat, and J. Sanabria, "Performance evaluation of container-based virtualization for high performance computing environments," *CoRR*, vol. abs/1709.10140, 2017. [Online]. Available: http://arxiv.org/abs/1709.10140

[4] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 233–240.

[5] D. Brayford and S. Vallecorsa, "Deploying scientific al networks at petaflop scale on secure large scale hpc production systems with containers," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, ser. PASC '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3394277.3401850

[6] Y. Wang, R. T. Evans, and L. Huang, "Performant container support for hpc applications," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3332226

[7] "Charliecloud Documentation," https://hpc.github.io/charliecloud/install.html.

[8] G. Slavova. Introducing Intel® MPI Benchmarks. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-mpi-benchmarks.html

[9] Blue Waters User Portal — SPP Benchmarks. [Online]. Available: https://bluewaters.ncsa.illinois.edu/spp-benchmarks