Container technologies on HPC applications

Guilherme Rezende Alles[1], **Alexandre Carissimi**[1], **Lucas Schnorr**[1]

[1]Instituto de Informatica – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

Abstract. *This paper explores the usage of container technologies to solve two relevant problems in high performance computing (HPC) applications. The first problem is reproducibility, expressed by the difficulty of precisely reproducing an execution environment to verify, validate or build on previous work. Because of the amount of potential dependencies and external variables an application depends upon, it can be extremely difficult to simulate the same software environment to reproduce previous research. The second problem is derived from the fact that HPC resources (e.g. clusters and grids) usually have very strict usage policies, that do not allow users to install arbitrary software to run their applications.*

Introduction

One fundamental problem of scientific computing research is that, in order for it to be credible, it should be reproducible. This means that the researcher has the responsibility of providing its audience with the resources necessary to run experiments and obtain similar results to the ones provided by the research. Although the concept of reproducibility is well known in the scientific community, it is very hard to put it to practice because of the large amount of variables that comprise an execution environment. Additionally, HPC resources are usually managed under strict usage policies, many of which do not allow the user to customize the environment with an arbitrary software stack (e.g. compilers and device drivers), thus making it very difficult to obtain an uniform execution platform across different clusters.

Although presented as difficult to deal with, these problems can be solved through the use of traditional hardware virtualization. By abstracting the hardware layer and providing a general, well known API to the software stack, users can run their own operating system and execution environment on top of a hypervisor, ensuring that the software stack will always be the same regardless of any hardware change. The problem with this approach is that, when considering HPC applications, the performance overhead introduced by the hypervisor is too high for this solution to be considered feasible.

Lately, container technologies have gained a lot of attention in the software development industry. This is mainly because containers are presented as a solution to package enterprise software in a controlled, static environment that will run on a wide variety of Linux systems. When compared to traditional virtual machines, containers have the advantage of not needing a hypervisor for virtualization, yielding performance extremely close to native but still providing application and software stack isolation.

By using containers to isolate and package HPC applications, we expect to encourage users to use containers to create reproducible, user configurable environments for their applications, thus diminishing the hassle of managing application dependencies and software stack differences when executing experiments across multiple HPC clusters.

In fact, this approach has already been used to a great extent in the software development industry. Container technologies such as Docker are used daily to deploy software microservices to cloud environments, and cloud providers such as AWS and Google Cloud Platform offer services to manage applications based on containers. In fact, Google announced in 2014 that all of its services run on top of containers, adding to more than 2 billion containers launching every day.

Considering the large adoption of containers in software development, we compare and contrast two container technologies, Docker and Singularity, with respect to their performance overhead and ability to solve common HPC workflow problems. For the performance overhead measurements, we will consider the same experiments running on a native environment as the baseline.

Related Work

In [REF], the author compares the compute performance of Docker containers to virtual machines, concluding that the former has a considerably lower overhead, when compared to the latter. This study, however, only explores the performance of single node applications, and does not present information on how containers can scale to distributed environments.

For multi node computations, [REF, Nguyen] proposes the creation of an MPI cluster using Docker containers in a distributed environment. In this proposal, the containers are connected through an orchestrator called Docker Swarm, which is responsible for assigning names and providing network connectivity transparently between the containers. *Mention that no performance information is available in this study*.

In [REF, GMK], the author introduces Singularity. Singularity is a container system designed for scientific research workflows, and it strives to solve some drawbacks of using Docker containers in HPC. The author discusses how Docker was not designed for shared, multi-user environments (such as supercomputers and HPC centers) and thus presents significant security issues when used in this context. As a consequence, it is very hard to find HPC centers that allow users to use Docker containers. Singularity, on the other hand, was built with these problems in mind and solves them in order to make HPC containers accessible and is already accepted and used in many supercomputers around the world.

Background and Experimental Context

Background

Mention containers as means of achieving software level virtualization and, consequently, process isolation.

Configurable (and reproducible) software stacks through the use of container images

Explain the theoretically negligible overhead when compared to native processes because of Unix *namespaces* and *cgroups*.

Talk about Docker's virtualization model - in a nutshell, virtualize everything that is possible

Singularity's virtualization model - virtualize only the essentials.

Experimental Context and Workload Details

Talk about the hardware stack: Grid 5000, the cluster environment, kadeploy3

Talk about the software stack: Debian9, Docker, Singularity, Ondes3D, NAS, Ping pong

The container infrastructure for Docker was built with the cluster proposed by [REF, Nguyen]: containers were connected using Docker Swarm and an overlay network.

The container infrastructure for Singularity is pretty much the same as the one with native processes. The only difference is that instead of distributing the application binary, I distributed the container image.

Talk about the experimental design: how were the results obtained and interpreted?

Results

Show experiment plots (approx. 3 plots)

Discuss some conclusions derived from the results

Conclusion

Summarize what has been discussed: the original problem, why containers might be a relevant solution, which technologies were tested

By looking at the experiment results, what are the conclusions that can be taken with respect to the performance overhead introduced by containers?

All things considered, is it okay to use containers for the kinds of application we tested? If it is (yes, it is), then which technology is the best one? (Singularity).

Why is Docker not suitable for HPC environments and especially distributed MPI applications?

Future work

Working with GPUs and other devices such as InfiniBand

Acknowledgements