Containerizing HPC Applications with Singularity

Authors: Rengan Xu, Frank Han, Nishanth Dandapanthula.

HPC Innovation Lab. October 2017

Overview

In this blog, we will give an introduction to Singularity containers and how they should be used to containerize HPC applications. We run different deep learning frameworks with and without Singularity containers and show that there is no performance loss with Singularity containers. We also show that Singularity can be easily used to run MPI applications.

Introduction to Singularity

Singularity is a container system developed by Lawrence Berkeley Lab to provide container technology like Docker for High Performance Computing (HPC). It wraps applications into an isolated virtual environment to simplify application deployment. Unlike virtual machines, the container does not have a virtual hardware layer and its own Linux kernel inside the host OS. It is just sandboxing the environment; therefore, the overhead and the performance loss are minimal. The goal of the container is reproducibility. The container has all environment and libraries an application needs to run, and it can be deployed anywhere so that anyone can reproduce the results the container creator generated for that application.

Besides Singularity, another popular container is Docker, which has been widely used for many applications. However, there are several reasons that Docker is not suitable for an HPC environment. The following are various reasons that we choose Singularity rather than Docker:

- **Security concern**. Because of Docker daemon, a user inside the Docker container is able to obtain root access on the host and then may act maliciously on the supercomputing cluster. In contrast, Singularity solves this by running the container with the user's credentials. The access permissions of a user are the same both inside the container and outside the container. Thus, a non-root user cannot change anything outside of his/her permission.

- **HPC Scheduler**. Docker does not support any HPC job scheduler, but Singularity integrates seamlessly with all job schedulers including SLURM, Torque, SGE, etc.

- **GPU support.** Docker does not support GPU natively. Nvidia Docker is a GPU-enabled Docker container, but it preinstalls various software that a user may not need. Singularity is able to support GPUs natively. Users can install whatever CUDA version and software they want on the host which can be transparently passed to Singularity.

  MPI support. Docker does not support MPI natively. So if a user wants to use MPI with Docker, a MPI-enabled Docker needs to be developed. If a MPI-enabled Docker is available, the network stacks such as TCP and those needed by MPI are private to the container which makes Docker containers not suitable for more complicated networks like Infiniband. In Singularity, the user's environment is shared to the container seamlessly.

Challenges with Singularity in HPC and Workaround

Many HPC applications, especially deep learning applications, have deep library dependences and it is time consuming to figure out these dependences and debug build issues. Most deep learning frameworks are developed in Ubuntu but they need to be deployed to Red Hat Enterprise Linux. So it is beneficial to build those applications once in a container and then deploy them anywhere. The most important goal of Singularity is portability which means once a Singularity container is created, the container can be run on any system. However, so far it is still not easy to achieve this goal. Usually we build a container on our own laptop, a server, a cluster or a cloud, and then deploy that container on a server, a cluster or a cloud. When building a container, one challenge is in GPU-based systems which have GPU driver installed. If we choose to install GPU driver inside the container, but the driver version does not match the host GPU driver, then an error will occur. So the container should always use the host GPU driver. The next option is to bind the paths of the GPU driver binary file and libraries to the container so that these paths are visible to the container. However, if the container OS is different than the host OS, such binding may have problems. For instance, assume the container OS is Ubuntu while the host OS is RHEL, and on the host the GPU driver binaries are installed in /usr/bin and the driver libraries are installed in /usr/lib64. Note that the container OS also have /usr/bin and /usr/lib64; therefore, if we bind those paths from the host to the container, the other binaries and libraries inside the container may not work anymore because they may not be compatible in different Linux distributions. One workaround is to move all those driver related files to a central place that does not exist in the container and then bind that central place.

The second solution is to implement the above workaround inside the container so that the container can use those driver related files automatically. This feature has already been implemented in the development branch of Singularity repository. A user just need to use the option “--nv” when launching the container. However, based on our experience, a cluster usually installs GPU driver in a shared file system instead of the default local path on all nodes, and then Singularity is not able to find the GPU driver path if the driver is not installed in the default or common paths (e.g. /usr/bin, /usr/sbin, /bin, etc.). Even if the container is able to find the GPU driver and the corresponding driver libraries and we build the container successfully, if in the deployment system the host driver version is not new enough to support the GPU libraries which were linked to the application when building the container, then an error will occur. Because of the backward compatibility of GPU driver, the deployment system should keep the GPU driver up to date to ensure its libraries are equal to or newer than the GPU libraries that were used for building the container.

Another challenge is to use InfiniBand with the container because InfiniBand driver is kernel dependent. There is no issue if the container OS and host OS are the same or compatible. For instance, RHEL and Centos are compatible, and Debian and Ubuntu are compatible. But if these two OSs are not compatible, then it will have library compatibility issue if we let the container use the host InfiniBand driver and libraries. If we choose to install the InfiniBand driver inside the container, then the drivers in the container and the host are not compatible. The Singularity community is still trying hard to solve this InfiniBand issue. Our current solution is to make the container OS and host OS be compatible and let the container reuse the InfiniBand driver and libraries on the host.

Singularity on Single Node

To measure the performance impact of using Singularity container, we ran the neural network Inception-V3 with three deep learning frameworks: NV-Caffe, MXNet and TensorFlow. The test server is a Dell PowerEdge C4130 configuration G. We compared the training speed in images/sec with Singularity container and bare-metal (without container). The performance comparison is shown in Figure 1. As we can see, there is no overhead or performance penalty when using Singularity container.

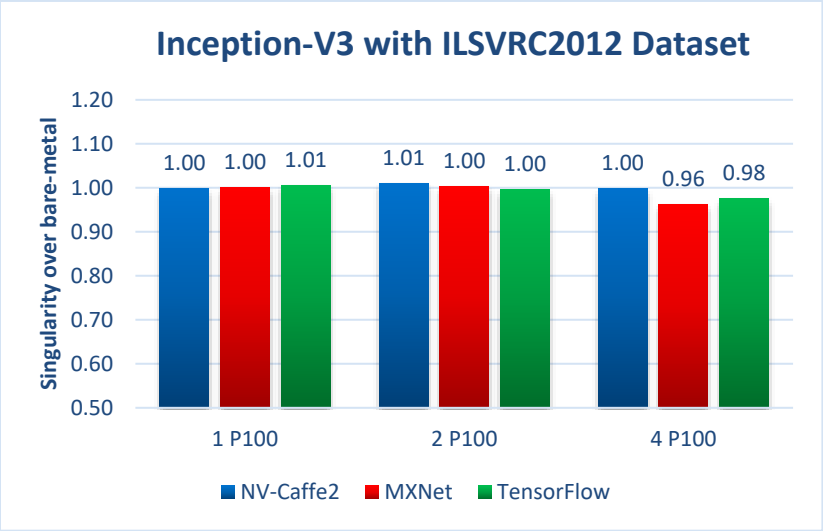

Figure 1: Performance comparison with and without Singularity

Singularity at Scale

We ran HPL across multiple nodes and compared the performance with and without container. All nodes are Dell PowerEdge C4130 configuration G with four P100-PCIe GPUs, and they are connected via Mellanox EDR InfiniBand. The result comparison is shown in Figure 2. As we can see, the percent performance difference is within ± 0.5%. This is within normal variation range since the HPL performance is slightly different in each run. This indicates that MPI applications such as HPL can be run at scale without performance loss with Singularity.

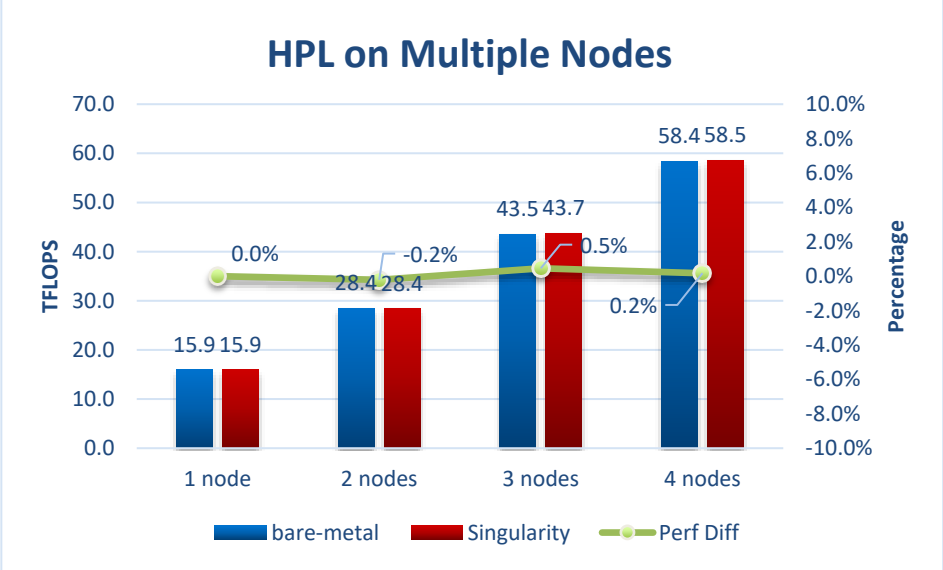

Figure 2: HPL performance on multiple nodes

Conclusions

In this blog, we introduced what Singularity is and how it can be used to containerize HPC applications. We discussed the benefits of using Singularity over Docker. We also mentioned the challenges of using Singularity in a cluster environment and the workarounds available. We compared the performance of bare metal vs Singularity container and the results indicated that there is no performance loss when using Singularity. We also showed that MPI applications can be run at scale without performance penalty with Singularity.