Performance Comparison of Multiple Containers Running Artificial Intelligence Applications

Minghui Sun, Ying Guo, hu Zhang, Wentai Cao and Meng Yuan
Qilu University of Technology (Shandong Academy of Sciences), Jinan 250000, China
Shandong Computer Science Center (National Supercomputer Center in Jinan), Jinan 250000, China
Shandong Provincial Key Laboratory of Computer Networks, Jinan 250000, China
1043119384@stu qlu.edu.cn

Abstract. Due to the performance overhead of virtual machines, traditional virtualization solutions are generally not suitable for artificial intelligence applications. Container-based virtualization technology represented by Docker can provide a customized environment, good portability, good compatibility and low overhead. In this article, we give a performance comparison among Docker, Singularity, and Charliecloud containers. In particular, CPU and GPU are used in each container to run a real application of artificial intelligence for the performance comparison. Finally, the experimental results are given and discussed.

1. Introduction
Due to the complexity of artificial intelligence applications and algorithms, there are many types of computing frameworks and algorithms, and they are evolving rapidly. It is a tedious task to configure and deploy different hardware environments and software applications for various artificial intelligence applications. The user deploys the corresponding library files and software based on their needs when using it. Because of the dependency between library files, upgrading some library files may break the compatibility between the library and another library, and attempting to use traditional deployment methods will result in low deployment efficiency and high operation and maintenance costs. The use of virtualization technology based on hypervisors (such as Xen, KVM) is an efficient way to solve the above problems, but the research of [1-3] shows that this method faces higher performance overhead.

Nowadays, lightweight virtualization technology is constantly developing and gradually mature, a container-based solution (such as LXC[4], Docker[5],Singularity[6] and Charliecloud[7]) is lightweight alternatives to hypervisors. Container technology provides a convenient packaging mechanism for applications. By encapsulating business applications, middleware, startup commands, etc. it ensures a high degree of consistency between the local environment and the deployment environment. This mechanism integrates applications and dependencies package into a single object, which has the advantages of being fast, efficient, and easy to migrate [8]. Unlike virtual machines, the container environment does not need to install an operating system and can run directly on the host operating system. The additional requirements of containers on system resources are much lower than virtual machines, and containers can save overhead by sharing the underlying resources of the host.[9][10] research shows that compared with traditional virtual machines based on hypervisors, containers have similar performance to native machines. From the perspective of the operating environment, the
container can run independently of hardware and software resources, with high compatibility. It can be packaged once and run on any machine with containers. In short, using containers can deploy applications quickly, reliably, and consistently.

However, with the continuous development and innovation of science and technology, computing power (CP) requirements in artificial intelligence applications continue to increase. In some cases, the CP power of the traditional CPU can no longer meet the needs of model training and inference, and more CP support of the GPU is needed. This also puts forward new requirements for us to run artificial intelligence applications in containers. The main contributions of this article are:

• Use nvidia-docker to support Docker to run artificial intelligence applications.
• Demonstrated the use of Charliecloud container to run artificial intelligence applications.
• Performance analysis of artificial intelligence applications using GPU training and prediction in the container.

The remaining parts of this article as follows:

• The second part introduces the various containers we will compare, summarizes some of the previous experience, and shows the difference between us and them.
• The third part describes the artificial intelligence applications used in containers.
• The fourth part introduces the experimental environment and specific experimental procedures.
• The fifth part discusses and presents our experimental results.
• The sixth part draws conclusions.

2. Related work

With the above advantages of container technology, container technology is now used by more and more people. The following will demonstrate the various container technologies we will examine.

2.1. Different containers

2.1.1. Docker.
Docker[4] is the most popular container platform and the standard for microservice deployment models. Each Docker container has its own process, which is started by the Docker Daemon. Docker containers are executed as root users by default. Docker is more suitable for service-type processes rather than job-type processes. At present, Docker can install the nvidia-docker plugin on the host to achieve the purpose of using the GPU in the container.

2.1.2. Singularity.
Singularity [5] began in 2015 as an open-source project of the Lawrence Berkeley National Laboratory, which was designed to solve the use of containers in high-performance computing. In terms of security, the user context always maintains the same when the Singularity container is started, the permissions are the same inside and outside the container, and there is no individual daemon. Singularity supports GPU by default, which can be configured when using the run command --nv parameter to use GPU.

2.1.3. Charliecloud.
CharlieCloud[6] is an open-source container technology developed by Los Alamos National Laboratory for supercomputing clusters. This technology provides a reproducible and unprivileged container workflow, using only 800 lines of code to help scientific computing users run workloads in supercomputing environments, avoiding most security risks, and implementing user-defined software stacks and effective isolation of the host operating system. CharlieCloud also supports GPU by default. After the user injects the executable files and libraries recommended by NVIDIA in the host into the Charliecloud image through the fromhost command, the user can use GPU computing in Charliecloud.
2.1.4. **Shifter.**

Shifter[11] originated from the National Energy Research Scientific Computing Center and high-performance computing manufacturer Cray project to support the deployment of computing applications bundled as Docker containers in the job scheduling system. The focus of its research is to use containers as a way to package the application running environment. Shifter uses its own image format, and both Docker images and VMs (virtual machines) will be converted to this image format during use. It turns out that Shifter is good at managing high-performance computing workflows, but to use image in Shifter, users must submit the image to the ROOT-controlled mirroring gateway through the RESTful API. And the fly in the ointment is that its maintenance work is not perfect, so it is not placed in our comparison queue.

2.2. **relevant research**

Predecessors have done a lot of research on the use of container technology. The research of [9][10] compared the performance of a variety of containers, using benchmarks to test the performance of the CPU, storage, and network in the container. However, these experiments did not use specific applications to compare the performance differences of the containers. In terms of using container technology to run artificial intelligence applications, the research of [12] compared the performance of singularity containers and bare metal machines using CPU and GPU to run artificial intelligence application training and prediction processes. The results show that Singularity containers are used in the training process will increase the running time by 33%. Using the Singularity container in the prediction process can decrease the running time by up to 25%, and the singularity container will not change the performance of the machine learning algorithm. The experiment of [13] compared the performance of OpenStack virtual machine, Docker container, and Singularity container using CPU in the training and prediction process of artificial intelligence applications. The results show that training and predicting artificial intelligence applications in a container is better than running in a virtual machine because the hypervisor of the virtual machine will cause additional overheads. For Docker and Singularity containers, the performance of Singularity is slightly higher than that of Docker when using the CPU to run artificial intelligence applications. The experiment of [14] designed an online artificial intelligence laboratory using a Docker cluster based on a delay queue and executed artificial intelligence experiments through remote access, which lowered the threshold of learning and increased the enthusiasm of students.

Compared with the experiment in [12], this article focuses on comparing the performance of different container technologies (Docker, Singularity, Charliecloud) in the training and prediction phases of artificial intelligence applications. Compared with [13], the performance of artificial intelligence applications using GPU training and prediction is compared in the containers that compare queues in this article. For [14], this article compares a variety of containers, which can help them select the appropriate container technology for the development of artificial intelligence application online laboratories.

3. **Description of artificial intelligence applications**

This article decides to use the recognition MINST handwritten data set as an application to evaluate the performance of different containers. The realization of this application demands two key tasks, namely the model training task and the model inference task. Model training is the task of calculating appropriate weights for the neural network. Model inference is a task that uses trained neural network to predict the results of input parameters [15].

The experimental test cases in this article use a general deep learning framework Tensorflow [16] to write a series of Python programs to implement the use of CPU and GPU to train the neural network and use the trained neural network to perform model inference and feedback the results. In the model training phase, we will test two kinds of neural networks. The first is called a small convolutional neural network (SCNN), as shown in Listing 1. The second is called a large convolutional neural network...
(LCNN), as shown in Listing 2. These lists are generated by calling summary() function from Keras [17].

Both of these networks consist of the same components essentially, including a 2D Convolutional Layer, an Average pooling Layer, Flattening Layer, and a Dropout Layer. The main difference between them is that LCNN has more layers than SCNN. SCNN has 8 layers, while LNN has 16 layers. Among them, SNN has 3 Convolutional layers, and LNN has 4 Convolutional layers. Each convolutional layer performs $3 \times 3$ convolutions. SNN has no Dropout layer, and LNN has 2 Dropout layers. All in all, when using LCNN, the additional layers will increase the computational load of model training and inference.

In the model inference phase, we will use the models trained by the CPU and GPU to perform inference operations on the MNIST handwritten data set.

In summary, we use these networks to test the following four workloads:

- Training, using CPU.
- Training, using GPU.
- Inference, using CPU.
- Inference, using GPU.

Through this division, we can see the impact of CPU-based or GPU-based processing load on Docker, Singularity, and Charliecloud. In the training phase, we will use the CPU and GPU respectively to perform network training operations. In the whole training process, there are 60,000 28*28-pixel pictures for training. For both LCNN and SCNN, it will be iterated 10 times. The trained network can distinguish 10 output categories, namely numbers 0-9. In the inference phase, the CPU and GPU used to perform inference operations on 10,000 pictures using the above mentioned trained mode.

| Layer (type) | Output Shape | Param # |
|--------------|--------------|---------|
| conv2d_1 (Conv2D) | (None, 26, 26, 6) | 60 |
| average_pooling2d_1 (Average) | (None, 13, 13, 6) | 0 |
| conv2d_2 (Conv2D) | (None, 11, 11, 16) | 880 |
| average_pooling2d_2 (Average) | (None, 5, 5, 16) | 0 |
| conv2d_3 (Conv2D) | (None, 3, 3, 120) | 17400 |
| flatten_1 (Flatten) | (None, 1080) | 0 |
| dense_1 (Dense) | (None, 84) | 90804 |
| dense_2 (Dense) | (None, 10) | 850 |

Total params: 109,994
Trainable params: 109,994
Non-trainable params: 0

Listing 1 SCNN Summary.

| Layer (type) | Output Shape | Param # |
|--------------|--------------|---------|
| conv2d_1 (Conv2D) | (None, 26, 26, 6) | 60 |
| conv2d_2 (Conv2D) | (None, 24, 24, 16) | 880 |
| zero_padding2d_1 (ZeroPadding2D) | (None, 26, 26, 16) | 0 |
| dropout_1 (Dropout) | (None, 26, 26, 16) | 0 |
| zero_padding2d_2 (ZeroPadding2D) | (None, 26, 26, 16) | 0 |
| conv2d_3 (Conv2D) | (None, 26, 26, 16) | 2320 |
| zero_padding2d_3 (ZeroPadding2D) | (None, 28, 28, 16) | 0 |
| average_pooling2d_1 (Average) | (None, 14, 14, 16) | 0 |
| zero_padding2d_4 (ZeroPadding2D) | (None, 18, 18, 16) | 0 |
| average_pooling2d_2 (Average) | (None, 9, 9, 16) | 0 |
| dropout_2 (Dropout) | (None, 9, 9, 16) | 0 |
| zero_padding2d_5 (ZeroPadding2D) | (None, 13, 13, 16) | 0 |
| conv2d_4 (Conv2D) | (None, 11, 11, 120) | 17400 |
| flatten_1 (Flatten) | (None, 14520) | 0 |
| dense_1 (Dense) | (None, 84) | 1219764 |
| dense_2 (Dense) | (None, 10) | 850 |

Total params: 1,241,274
Trainable params: 1,241,274
Non-trainable params: 0

Listing 2 LCNN Summary.
4. Description of artificial intelligence applications
In this paper, we use Quan Cloud platform as our experimental environment to test the performance of Section 3 neural network execution using CPU and GPU in Docker, Singularity and Charliecloud. The configuration of the server is shown in Table 1 and the version of the installation container is shown in Table 2.

| Table 1   | Server configuration          |
|-----------|-------------------------------|
| OS        | Ubuntu 16.04                  |
| Linux Kernel | 4.4.0-146-generic            |
| RAM       | 8G                            |
| CPU       | 4 core Intel Xeon E312xx      |
| GPU       | NVIDIA TITAN Xp               |

| Table 2   | Container version           |
|-----------|-----------------------------|
| Docker    | 20.10.6                     |
| Singularity | 3.0.3-1                   |
| Charliecloud | v0.23                    |

The container platform in this experiment is divided into CPU version and GPU version. The CPU version of the Docker container image is created by the build command using Dockerfile, and the Dockerfile recipe can be obtained from Listing 3. After the Docker image is created, execute the build and ch-builder2tar commands to convert it into a Singularity image and Charliecloud image. The three CPU versions of the container have been created in this way.

```bash
From centos
LABEL maintainer="1043119384@stu.slu.edu.cn"

# Show installation using yum
RUN yum -y update && yum install -y git wget python3 python3-pip

# Show installation base library package
RUN pip3 install --upgrade pip
RUN pip3 install numpy==1.15.4
RUN pip3 install ipython
RUN pip3 install scikit-learn
RUN pip3 install scikit-image
RUN pip3 install jinja2
RUN pip3 install imageio
RUN pip3 install h5py==2.10.0
RUN pip3 install pyyaml
RUN pip3 install pandas
RUN pip3 install keras==2.1.0

# Show the installation of the machine learning framework
RUN pip3 install tensorflow==1.12.0

# Copy the code into the container
RUN mkdir AI
COPY AI /AI
```

Listing 3 Docker Recipe.
In order to support the use of GPU to train neural networks in the container, the NVIDIA driver needs to be installed on the server. CUDA 10.1 was installed on the server in this experiment. To make the GPU version of the Docker image, users need to deploy the nvidia-docker2 plug-in on the host, and install the NVIDIA driver of the same version as the host and the library files in the above Dockerfile in the container image. It should be mentioned here that the machine learning framework is installed with Tensorflow-gpu, and the GPU version of the Docker image is acquired after the build. The GPU version of the singularity image and the Charliecloud image are also converted through the Docker image.

For the above six types of tests, we require exclusive access to the node in the experiment to ensure that other users and processes will not affect the results from the perspective of resource utilization or running time. Then execute each test 20 times on this node, and record a series of performance indicators of the training model and model inference in different containers for our subsequent comparison. Finally, the performance of the container platform is judged by comparing the time consumed by different container model training tasks and model inference tasks.

5. Experiment results

5.1. Model training stage

Figure 1 summarizes the time spent in the model training phase of each type of experiment.

![Graphs depicting training times for different models and containers.](image)

It can be seen that the training time of using CPU to train LCNN and SCNN in different containers is relatively close, and there is no obvious difference. But, there is a difference between using GPU to train LCNN and SCNN in the container. Whether it is SCNN or LCNN, the time to train the network using GPU in Singularity container is shorter than that in Docker and Charliecloud containers. This is a very unique point. If you train for a long period of days or weeks, using Singularity containers to train neural networks can reduce an army of time. The reason for the slowness of Docker containers may be that the nvidia-docker2 plug-in brings some additional overhead. The reason for this phenomenon in the Charliecloud container may be that the NVIDIA library and executable files injected by the host have an impact on the runtime.
5.2. Model inference stage

Table 3 and Table 4 summarize the time spent in the model prediction phase for each type of experiment.

**Table 3  Results for SCNN inference.**

|                      | Mean(us) | Min(us) | Max(us) | STD   | Accuracy   |
|----------------------|----------|---------|---------|--------|------------|
| Docker inference (CPU) | 1700     | 1606    | 2002    | 108    | 99.96%     |
| SingularityInference (CPU) | 1641     | 1547    | 1712    | 64     | 99.01%     |
| Charliecloud Inference (CPU) | 1668     | 1528    | 1744    | 65     | 99.03%     |
| Docker inference (GPU)  | 893      | 760     | 1075    | 92     | 99.03%     |
| Singularity Inference (GPU) | 890      | 804     | 1117    | 104    | 99.04%     |
| Charliecloud Inference (GPU) | 881      | 740     | 1032    | 94     | 99.07%     |

**Table 4  Results for LCNN inference.**

|                      | Mean(us) | Min(us) | Max(us) | STD(us) | Accuracy   |
|----------------------|----------|---------|---------|---------|------------|
| Docker inference (CPU) | 8891     | 8694    | 9134    | 121     | 99.06%     |
| Singularity Inference (CPU) | 8686     | 8482    | 8965    | 141     | 99.08%     |
| Charliecloud Inference (CPU) | 8700     | 8456    | 8945    | 143     | 99.11%     |
| Docker inference (GPU)  | 1033     | 905     | 1185    | 64      | 99.05%     |
| Singularity Inference (GPU) | 1021     | 860     | 1150    | 83      | 99.17%     |
| Charliecloud Inference (GPU) | 1039     | 807     | 1175    | 166     | 99.15%     |

It can be seen that although the time consumption of different containers in the model inference phase is very small, the performance shown by singularity is still better than that of Docker and Charliecloud. In addition, in this test environment, there is a point worthy of attention. After statistics, the STD of the GPU version of Docker is lower than the other two containers. Docker has shown good stability in the prediction process and can perform prediction tasks with relatively stable performance. Although Docker's performance is not the best, it has the best stability. From the perspective of the accuracy of model inference, the inference accuracy between various containers is relatively close. Which container is used to train the neural network will not affect the accuracy of the network. Therefore, using the container to train the neural network can acquire a consistent model, and will not affect the performance of the neural network.

6. Conclusions

In this article, we show a comparison of the CPU and GPU performance of three containers (Docker, Singularity, and Charliecloud) executes artificial intelligence applications. According to the results of the experiment, both the CPU version and the GPU version of Singularity has better performance than other containers in the model training phase. The reason for this may be that the other two containers have excessive overhead in GPU support. In the stage of model inference, Singularity containers have better performance using CPU or GPU, but Docker containers appear to have better performance in terms of performance stability. From the perspective of the accuracy of the model, which container is used to train the neural network will not impact the accuracy of the network, so using the container to train the neural network can get a consistent model without affecting the performance of the neural network.

Due to the consistency, compatibility, portability, and other advantages of container technology, when containers need to be used to run artificial intelligence applications in a production environment, we recommend using the GPU version of the Singularity container during the model training. In the model inference, we suggest user select Singularity container with better performance or Docker container with higher stability and widely used.

To solve the problems faced by artificial intelligence application deployment and operation and maintenance, this article evaluated the performance of using CPU and GPU to run artificial intelligence applications in a variety of containers and achieved good results. Each container technology tested has
a good output. In the future, we hope to use parallel GPUs in containers to execute machine learning applications.

Acknowledgments
This research was supported by grants from the Shandong Provincial Natural Science Foundation (ZR2019QF014).

References
[1] Felter W, Ferreira A, Rajamony R and Rubio J 2015 An Updated Performance Comparison of Virtual Machines and Linux Containers 15th IEEE Int. Symposium on Performance Analysis of Systems and Software (United States: Philadelphia) pp 171-172
[2] Morabito, R, Kjällman, and J, Komu, M 2015 Hypervisors vs. Lightweight Virtualization: a Performance Comparison 2015 IEEE Int. Conf. on Cloud Engineering (United States: Tempe) pp 386-393
[3] Li Z, Kihl M, Lu Q and Andersson J A 2017 Performance overhead comparison between hypervisor and container based virtualization SC 2017: The International Conference on High Performance Computing, Networking, Storage and Analysis (Taiwan: Tamkang University Taipei) pp 955-962
[4] Moravcik M, Segec P, Kontsek M, Uramova J and Papan J 2020 Comparison of LXC and Docker Technologies 18th IEEE Int. Conf. on Emerging eLearning Technologies and Applications (Slovakia: Kosice) pp 481-6
[5] D Merkel 2014 Docker: Lightweight linux containers for consistent development and deployment Linux Journal 2014 239
[6] Kurtzer G.M, Sochat V and Bauer M.W 2017 Singularity: Scientific containers for mobility of compute PLOS ONE 12 5
[7] Friedhorsky R and Randles T 2017 Charliecloud: unprivileged containers for user-defined software stacks in hpc Int. Conf. for High Performance Computing, Networking, Storage and Analysis, SC 2017 (United States: Denver)
[8] Boettiger, C 2014 An introduction to Docker for reproducible research 8th Workshop on Large-Scale Distributed Systems and Middleware (United Kingdom: Microsoft Research Cambridge21 Station Road Cambridge) pp 71-79
[9] Kovács, Á 2017 Comparison of different Linux containers 40th Int. Conf. on Telecommunications and Signal Processing (Spain: Hotel SB Diagonal Zero Superior, Placa Llevant) pp 47-51
[10] Sande Veiga V, Simon M, Azab A, Muscianisi G, Fernandez C, Fiameni G and Marocchi S 2019 Evaluation and Benchmarking of Singularity MPI containers on EU Research e-Infrastructure 1st IEEE/ACM Int. Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (United States: Denver) pp 1-10
[11] Gerhardt L, Bhimji W, Bhimji W, Fasel M, Jacobsen D, Mustafa M, Porter J and Tsulaia V 2017 Shifter: Containers for HPC 22nd Int. Conf. on Computing in High Energy and Nuclear Physics (United States: San Francisco) 898 2
[12] Jordan B R, Barrett D, Burke D, Jardin P, Littrell A, Monticciolo P, Newey M, Piou J and Warner K 2019 Singularity for machine learning applications - Analysis of performance impact 2019 IEEE High Performance Extreme Computing Conf. (United States: Westin Hotel Waltham)
[13] Marquez Jack D and Castillo Mario 2021 Performance Comparison: Virtual Machines and Containers Running Artificial Intelligence Applications Information Technology and Systems - ICTIS 2021 (China: Shanghai) pp 199-209
[14] Liu B, Huang Y, Huang T, Feng W, Zhong G 2021 Artificial intelligence online laboratory based on docker cluster and delay queue Int. Conf. on Applications and Techniques in Cyber Intelligence, ATCI 2020 (China: Fuyang) pp 307-312
[15] Copeland M 2016 What’s the Difference Between Deep Learning Training and Inference? https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai
[16] Abadi M et al. 2016 TensorFlow: A system for large-scale machine learning *12th USENIX Symposium on Operating Systems Design and Implementation* (United States: Savannah) pp 265-83

[17] tf.keras.models.Model *TensorFlow*
https://www.tensorflow.org/api_docs/python/tf/keras/models/Model