ARC containers for AI workloads

Singularity performance overhead

Marvin Newlin
Air Force Institute of Technology
Electrical and Computer Engineering
Wright-Patterson AFB, OH, United States
marvin.newlin@afit.edu

Kyle Smathers
Air Force Institute of Technology
Electrical and Computer Engineering
Wright-Patterson AFB, OH, United States
kyle.smathers@afit.edu

Mark E. DeYoung
Air Force Institute of Technology
Electrical and Computer Engineering
Wright-Patterson AFB, OH, United States
mark.deyoung@afit.edu

ABSTRACT

Containerization has taken the software world by storm. Deployment complications, like requiring elevated (i.e. "root") permissions to run, have slowed the adoption of containers in shared advanced research computing (ARC) environments. Singularity is a containerization approach that is designed for ARC in shared high performance computing (HPC) clusters. With the creation of the Singularity, there is finally a viable scientific container solution. However very few papers have looked at the performance trade-offs of deploying applications using a container based model. The authors are not aware of any published studies evaluating the trade-offs of the deployment models with complex Artificial Intelligence (AI) workloads. Without detailed evaluations of the performance trade-offs scientists and engineers are unable to make an informed decision on deployment model for time sensitive training or low power inference. Furthering previous research in this area and using emerging community developed benchmarks, we examine performance trade-offs of running AI workloads in a containerized Singularity environment.

CCS CONCEPTS

• **Information systems** → *Computing platforms*; • **Computing methodologies** → Distributed computing methodologies; • **Software and its engineering**;

KEYWORDS

Advanced Research Computing, High Performance Computing, Artificial Intelligence, Machine Learning, Deep Learning

ACM Reference format:
Marvin Newlin, Kyle Smathers, and Mark E. DeYoung. 2019. ARC containers for AI workloads. In *Proceedings of PEARC19, Chicago, IL, USA, July 28 - August 1, 2019,* 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this paper we utilize the emerging MLPerf benchmark [2] to evaluate the performance difference between two application deployment models: process based and container based. The contribution of our work is an evaluation of ARC containers for AI workloads that characterizes Singularity container performance overhead.

This paper is organized as follows. In section 2 we present an overview of containerization technology. In section 3 we describe previous works in this area and briefly discuss the benchmarks we use. Next, we describe our implementation in section 4. In section 5, we evaluate the performance of containerized benchmark tasks vs. running as a process. Finally, we provide conclusions and propose future work in section 6.

2 BACKGROUND

Container based deployment models have taken the software development and application deployment world by storm, partly thanks to its close ties with the "DevOps" movement. DevOps, a portmanteau of Development and Operations, is typically defined as "the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity." [3] Containerization is one of the "tools" that enables high velocity. A developer can rapidly prototype, build and then deploy a containerized application or service. Containers are appealing to the scientific computing community because they allow end users to provision runtime environments that are tuned for their workload. Container based deployment models can also support reproducibility of runtime environments.

Shared HPC clusters used in advanced research computing (ARC) often use process based runtime models. As shown in figure 1 the operating system mediates access between physical hardware and a runtime environment. Each application is isolated in a unique process but all applications share a common runtime environment. The runtime environment includes programming language support, pre-compiled binary libraries of common functionality, and file systems.

Deployment of applications in a virtual machine (VM) provides additional isolation between applications. Each application is provisioned with a complete operating system and unique runtime environment. While this enables user specific configurations it introduces additional layers with context switch cost. It can also drive over-provisioning of physical resources to accommodate resource fragmentation.

Application deployment using containers eliminates some of the overhead incurred by VM oriented deployments. Containers, unlike

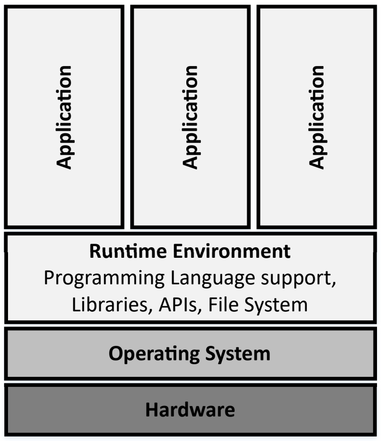

Figure 1: Process Deployment Model - an operating system can run multiple applications each with a unique process context. While at the same time providing a shared runtime environment.

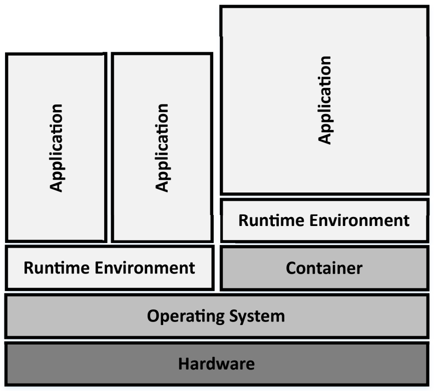

Figure 2: Container Deployment Model - native processes and contained processes are both managed by a single running operating system.

VMs, do not mask device topology which enables optimization for libraries such as Intel Math Kernel Library (MKL). As shown in figure 2, containers and native processes can co-exist, each with it's own runtime environment.

As an example of container popularity and scale, Amazon and their Amazon Web Services have met much of their success thanks to adopting a DevOps model and the associated containerization technology. Their Elastic Computer Cluster (EC2) is built on the premise of rapid scalability which helps them survive peak shopping seasons. [3] Netflix is another company that thrives on rapid scalability, essentially scaling their data center at will and on demand. [3][16] As an example Netflix surpassed 1 million containers launched and destroyed in one week back around April 2017, something that would be impossible if these were heavyweight VMs. As an example, if it takes 15 mins to install and configure a VM and it's operating system, 15 x 1M = 10,417 days or about the time it takes Saturn to orbit the Sun.[16]

Despite advances in the commercial sector, containerization is just beginning to catch on in the scientific community. [5] Prohibitions in the deployment of the mainstream container runtimes like Docker has delayed many large shared use ARC systems from utilizing containerization technology. For example, Docker containers running with elevated or "root" privileges cause grave security concerns. Despite these concerns, there have been previous attempts to investigate the performance trade-offs of using containers on scientific or CPU intense workloads. The initial estimate of these papers is to be expected, a small overhead was observed versus using "bare metal" installs. These slowdowns are generally eclipsed however by the benefits to experiment reproducibility, portability and integrity. By design, a container can be crafted for one scenario and then transplanted and run on many different physical environments. This finally enables a long sought after feature in many large shared systems: user defined software stacks. Code library and tooling versions can quickly get out of hand in shared environments causing all sorts of conflicts between end users and administrators. The separation provided by a container allows users to define the exact versions of all the supporting tooling and bring it along, without fear of conflicting with others runtime environment.

3 RELATED WORKS

Several efforts related to ours are more limited in scope only testing raw performance numbers i.e. (GFLOPS, MB/s) of containers vs. native. While valid tests, this left a gap of understanding of realistic AI workloads like deep learning (DL) and machine learning (ML). The focus of this paper therefore is to answer the question of how much, if any, overhead is incurred by container deployment models for complex DL/ML training and inference tasks. Because we are targeting ARC environments we specifically consider Singularity containers vs. native processes for AI workloads.

3.1 Container Overhead Tests

As mentioned previously, there are other works evaluating container overhead. However, these focused on more granular performance details like raw CPU performance with limited memory movement between nodes. [6] As an aside, Higgins et al. also showed a clear performance superiority over containers over entire VMs.[9] Miguel et al was one of the first papers to examine virtualization usage in HPC environments. [19]. They were a bit ahead of their time however and tested raw Linux Containers (LXC) which at the time were not very user friendly, not gaining traction until the Docker API came out to solve it. The paper seen at [15] was one of the better examples running benchmarks that involved multiple nodes and their interconnects. Despite their overall "mixed" findings they provided an excellent snapshot of the state of the art around 2015 and held hope that containers may be the future. As an example they found that the virtualized networking in earlier versions of the Linux kernel introduced a huge (180%) overhead compared to bare metal. Active work has been ongoing with the kernel and they show more recent kernels only saw about a 24% overhead.

One area that these container tests all fell short on however is diversity of hardware and software. All the tests ran were CPU bound on Intel based hardware either on premise or in a remote HPC environment. They all also used the Docker runtime which while the mainstream in many other sectors is not the only one available. Most modern complex Machine Leaning algorithms rely on either specialized hardware like the Google Tensor Processing Unit (TPU) [1] or they rely on the massively parallel power of modern Graphics Processing Units (GPU). With third party drivers and support Docker can utilize accelerator, like a GPU, however it further complicates the deployment. [13]

3.2 Science Focused Container Engines

Containerization, as it was originally built, was not very user friendly. It took around eight years after being introduced in the Linux kernel for open source API layers like Docker to open this technology for broad acceptance and commercialization. These tools enabled DevOps for single user or single administrator systems, however they were not designed for systems that either cannot be isolated or are shared.

Singularity was proposed in 2017 as a solution that provides a container engine that does not require elevated privileges and enables portability of containers. [11] For researchers, this perfect combination allows users on shared systems to deploy their own runtime environments without effecting other users on the same system. On top of all of this, the container is saved as a single SHA256 hashed file that can be moved, integrity checked and run (or reproduced) on any system that's running the engine.

Although it has a similar architecture to CharlieCloud and Shifter Singularity also enables access to componets like GPUs that are critical to run many ML and DL frameworks.[11][14] Additionaly, Singularity images are easily portable becasue they can be contained in a single file. This eases portability of the same image between many target systems and thus improves reproducibility of experiments.

One paper exists that examines any overhead of Singularity's performance with respect to message passing interface (MPI) workloads and provides a solid foundation to further research in this area.[20] Since MPI is seen as the defacto programming model in large parallel environments, it lines up nicely with the parallel nature of machine learning workloads. The authors also propose a four dimensional model of evaluation but it falls a little short on the diversity spectrum. Two ends of one dimension are two different flavors or Intel based chips, and another dimension is their associated two different RAM layouts. No accelerators like GPU were considered. Regardless throughout their extensive tests they saw no more than a 8% overhead on any of the dimensions.

4 IMPLEMENTATION

Our evaluation of ARC containers for AI workloads is implemented by running experiments on HPC compute nodes and ML/DL nodes described in section 4.1. We build application deploymets of AI workloads in compatibly configured native processes and Singularity containers. We use selected benchmarks from MLPerf, an emerging machine Learning benchmark effort.[2] One goal of MLPerf is building a common set of metrics that enables ML creators and consumers to measure system performance for both training and inference models. This broadly accepted set of rules and scores could help spur innovation and competition that ultimately benefits the entire ML/DL community. [2]

4.1 System Specifications

We used two types of HPC compute nodes and three types of ML/DL nodes to run experiments. The underlying hardware and operating systems used in our experiments are shown in table 1 and table 2. Compute capability indicates features provided by the GPU hardware[1]. The compute unified device architecture (CUDA) driver version determines which CUDA API version can be used in the runtime environment.

Table 1: System Specifications: HPC compute nodes

Configuration	Type-1	Type-2
CPU	1x Intel Xeon Platinum 8168	1x Intel Xeon E5-2699 v4
Accelerator	1x NVIDIA Tesla P100 (16 GB)	1x NVIDIA Tesla P100 (16 GB)
RAM	384 GB	256 GB
Network	Intel Omni-Path 100Gb	Cray Aries
Singularity	2.6	2.6
OS	RHEL 7.5	Cray Linux Environment
CUDA Driver	390.46	396.44
Compute Capability	6.0	6.0

Table 2: System Specifications: ML/DL nodes

Configuration	Type-3	Type-4	Type-5
CPU	2x Intel Xeon E5-2660 v3	1x Intel i7-5930K	1x Intel i7-6950X
Accelerator	8x NVIDIA Titan V (12 GB)	4x NVIDIA Titan X (12 GB)	4x NVIDIA GTX 1080Ti (12 GB)
RAM	755 GB	64 GB	128 GB
Network	Ethernet 10GbE	Ethernet 1GbE	Ethernet 1GbE
Singularity	2.6	2.6	2.6
OS	Ubuntu 16.04	Ubuntu 18.04	Ubuntu 18.04
CUDA Driver	415.27	415.27	415.27
Compute Capability	7.0	6.1	6.1

4.2 Runtime Environments

We use Conda[2] and GNU Environment Modules[3] to dynamically configure the runtime environment when deploying the benchmark

[1]CUDA GPUs - https://developer.nvidia.com/cuda-gpus
[2]Conda - https://conda.io
[3]GNU Environment Modules - https://sourceforge.net/projects/modules/

application as a native process. We build a compatible container image that uses Conda when deploying the benchmark application as a Singularity container. The same configuration of CUDA Device API, python, frameworks, and dependent libraries is used for the native and containerized benchmark. The operating system for each of the Singularity images is Ubuntu 16.04.

An equivalently configured native environment and Singularity image executed on the same type of hardware are considered a compatible runtime environment. It must be noted that the environments are not necessarily identical. We did not attempt fully reproducible builds at the bit-image level. Instead, compatibility is achieved by using the same API versions for frameworks. For the purposes of testing, we define a *compatible configuration* as similar hardware/software capabilities (e.g. HPC with a single P100, using the CUDA 9 API). Ideally, the only difference in the comparison of runtimes is the native process versus Singularity container.

In general, we use CUDA version 9.0 and 9.1. An additional requirement for many ML/DL frameworks is the NVIDIA CUDA Deep Neural Network library (cuDNN)[4]. We use a version of cuDNN compatible with the CUDA API required by the framework used by the benchmark under test.

4.3 Benchmarks

Machine learning and deep learning techniques are applied to a wide variety of tasks. Some workloads include image classification, language translation, and product recommendation. Because of the variety we need a common metric to characterize benchmark results. MLPerf and its cousin DAWNBENCH both use "Time to Accuracy" (TTA), the total time to train a model algorithm to a selected prediction accuracy. The TTA metric was further evaluated by Coleman who found despite the stochastic nature of many DL and ML algorithms that the metric remains valid. They also validate that models "tuned" to perform to this metric do not sacrifice real work performance to beat the test [5]. For our experiments we will utilize TTA as a primary measure of real-world overhead.

For the experiment, we ran each of the four selected benchmarks in the two deployment configurations (native and Singularity) on at least one type of HPC (both if possible) from table 1 and on the ML/DL nodes from table 2.

Sentiment Analysis - The Sentiment Analysis benchmark performs the classification task of sentiment analysis (positive or negative) based on the IMDB movie review dataset. [10][12] The dataset used for this task is relatively small and is not a very intensive task, but is useful as a baseline. The measure of accuracy for this benchmark is the percentage of correct classifications made on the test dataset. Thus, the TTA metric measure the time in seconds that it takes for the benchmark to reach the target accuracy. The environment for this benchmark consists of PaddlePaddle, CUDA 9.0, and cuDNN 7.

RNN Translator - The RNN Translator benchmark performs the task of translating news articles from German to English using the WMT16 dataset via a Recurrent Neural Network (RNN). [18] [4] The measure of accuracy for the benchmark is the BLEU score, a standardized translation metric. For the RNN Translator benchmark, the target BLEU score is 25, so the TTA metric measures the amount of time it takes for the RNN Translator benchmark to reach a BLEU score of 25. The environment for this benchmark consists of PyTorch 1.0, CUDA 9.1, and cuDNN 7.

Translation - The Translation benchmark performs translation from German to English on the WMT16 dataset, the same dataset used for the RNN Translator benchmark. With the Translation benchmark, the translation is performed by attention mechanisms rather than with an RNN. [17] [4] Like the RNN version, Translation also uses the BLEU score for evaluation and is coded to stop when it reaches a BLEU score of 25. Thus, the TTA metric measures the amount of time for the Translation benchmark to converge to a BLEU score of 25. The environment for this benchmark consists of Tensorflow 1.9.0, CUDA 9.0 and cuDNN 7.

Recommendation - The Recommendation benchmark uses the MovieLens 20 Million dataset to perform the classification task of recommendation. The model is trained on a binary set for whether or not a user interacted with a specific item and accomplishes this through the use of a Neural Collaborative Filter. [8][7] The accuracy measure for this benchmark is the hit rate at 10 for 999 negative items, so the TTA measures the amount of time to reach this measure. The environment for this benchmark consists of PyTorch v1.0, CUDA 9.1, and cuDNN 7.

5 EVALUATION

For hypothesis tests our goal was to collect results from at least 30 experimental runs for each compatible configuration. To conduct hypothesis tests, we ran the results of the native process runs and the Singularity runs through an independent two-tailed t-test to test for statistical significance.

5.1 Evaluation Criteria

Prior to performing our evaluation on the results of the MLPerf benchmarks we established the null and alternative hypotheses for performing hypothesis tests on the runtime results. The null hypothesis, H_0 is:

H_0: There is no statistically significant difference in mean runtime between native process and within the container. i.e. Native runtime = Singularity runtime.
Naturally, the alternative hypothesis is:
H_A: There is a difference in the mean runtime between native process and within the container. i.e. Native runtime $\neq$ Singularity runtime. In this case, we use $\neq$ to mean a statistically significant difference.
To evaluate our data, we used two-tailed independent T-tests on the runtimes of compatible configurations. We chose a 95% confidence interval for the tests (p-value = 0.05) as that is a standard choice for p-value.

5.2 Benchmark Result Evaluation

Sentiment Analysis Initially, the target accuracy for this benchmark was hard-coded to 90.56 in the code, but it would rarely ever converge to this target accuracy, thus invalidating the TTA metric. To fix this, we modified the target accuracy to 90.52 and with this slight modification we were able to get consistent convergence, thus letting us accurately measure TTA.

[4]NVIDIA cuDNN - https://developer.nvidia.com/cudnn

We ran the sentiment analysis benchmark in the HPC environment on a single P100 GPU in the Type-1 and Type-2 configurations. The sentiment analysis benchmark only uses a single GPU even if multiple GPUs are available, due to its reliance on the PaddlePaddle library, which only supports a single GPU configuration. We collected 100 Sentiment Analysis runtimes and then randomly sampled 30 for the t-test. A histogram of the runtimes is shown in figure 3.

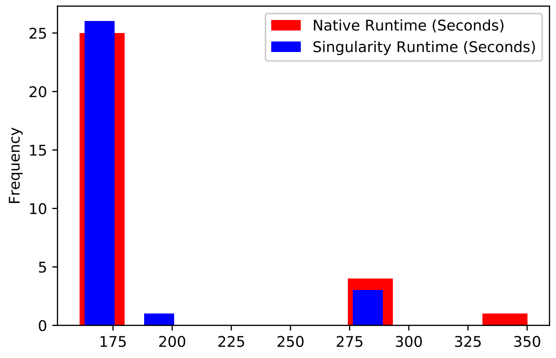

Figure 3: Histogram of runtime in seconds of Sentiment Analysis benchmark run on Type-1 (HPC node)

As seen in figure 3 and table 3 the two deployment models have similar performance. Since we are dealing with 30 samples of each, our degrees of freedom for the T-table was 58, yielding a critical T-value of 2.0.

Table 3: Sentiment Analysis Type-1 (HPC node) Results

Statistic (seconds)	Native	Singularity
Mean	174.30	178.63
Standard Deviation	39.143	45.322
Minimum Value	161.00	163.00
Maximum Value	296.00	350.00

Based on this data, the result of the t-test was a t-statistic of -0.39633 and a p-value of 0.69331. Since the t-statistic was not greater than out critical t-value of 2.0 and the p-value was not lower than our threshold of 0.05, we fail to reject H_0 for the sentiment analysis benchmark.

On the the Type-3, 4, and 5 configuration (ML/DL node) whose specifications are listed in table 2, we also ran the Sentiment Analysis benchmark 100 times for each deployment model. We again randomly sampled 30 of the 100 runtimes. A a histogram of the Type-3 (ML/DL node) runtimes is shown in figure 4.

The results, in table 4, show experiments on ML/DL nodes have longer mean runtimes than the HPC nodes. The longer runtime is expected because the GPUs in the ML/DL nodes have less memory as shown in table 1 and table 2. Additionally, benchmark data files are on shared storage accessed over the network connection shown

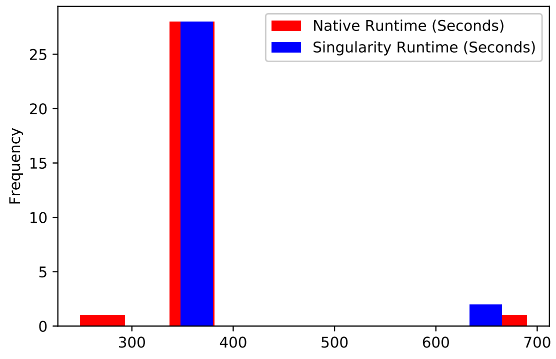

Figure 4: Histogram of runtime in seconds of Sentiment Analysis benchmark run on Type-3 (ML/DL node)

in table 1 and table 2. Each HPC cluster provides a locally mounted Lustre parallel file system which is accessible from login and compute nodes. The ML/DL nodes use network attached storage with NFS.

Table 4: Sentiment Analysis Type-3 (ML/DL node) Results

Statistic (seconds)	Native	Singularity
Mean	358.17	374.93
Standard Deviation	65.458	78.637
Minimum Value	249.00	348.00
Maximum Value	690.00	665.00

As with the HPC environment experiments, our critical t-value is 2.0 and our threshold p-value is 0.05. The resulting t-statistic was -0.89757 with a p-value of 0.37313. Again, the t-statistic is below the critical t-value and the resulting p-value is not below 0.05 so we fail to reject H_0 in this case as well.

Translation - Recurrent (RNN Translator) RNN Translator is a a data parallel workload and will use all available GPUs. The ML/DL nodes provide multiple GPUs while the HPC nodes do not. Because of this, we reduced the number of samples collected on HPC nodes due to long runtimes.

The results of the RNN Translator benchmark for Type-1 (HPC node) are shown in figure 5. The results are similar to the other benchmarks in that the distributions are fairly close to one another and the values in table 5 confirm what we see in the histogram. The mean runtimes are nearly identical to one another.

Due to queuing restrictions and wait times in the HPC environment, we only ran 20 RNN Translator experiments for Type-1 and Type-2 configurations. For the independent two-tailed t-test on these values, we have a p-value of 0.005 and a critical t-value of 2.021. The results of the t-test yields a t-value of 0.1399 and a p-value of 0.889. Thus, since both values do not meet the thresholds, we fail to reject the null hypothesis in this case.

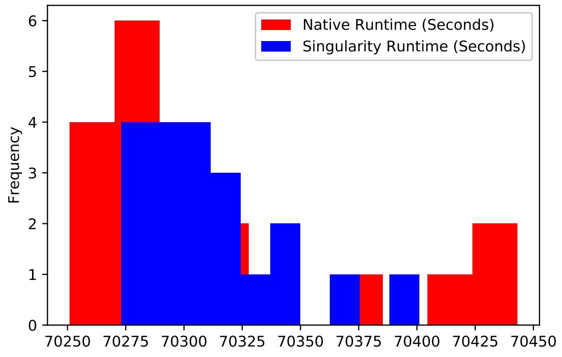

Figure 5: Histogram of runtime in seconds of RNN Translator benchmark run on Type-1 (HPC node)

Table 5: RNN Translator Type-1 (HPC node) Results

Statistic (seconds)	Native	Singularity
Mean	70314.0	70311.9
Standard Deviation	59.148	11.287
Minimum Value	70251.0	70273.0
Maximum Value	70443.0	70401.0

To examine a multiple GPU configuration, we ran 27 runs of RNN Translator on Type-3 (ML/DL node) from table 2. A histogram of the runtime results is shown below in figure 6.

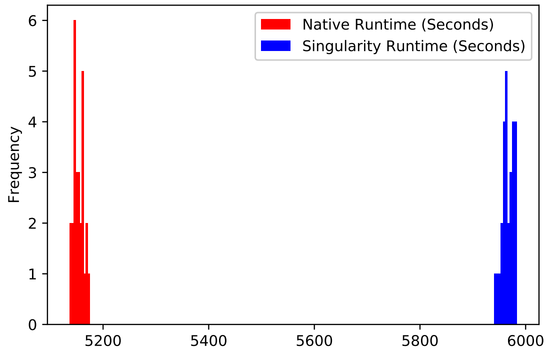

Figure 6: Histogram of runtime in seconds of RNN Translator benchmark run on Type-3 (ML/DL node)

The RNN Translator runs from the multiple GPU configuration provide drastically different results than the Sentiment Analysis benchmark and the single GPU RNN Translator results, and we can actually see a difference in the mean runtimes of the two configurations with this benchmark. The statistics of the runtimes are shown in table 6.

Table 6: RNN Translator Type-3 (ML/DL node) Results

Statistic (seconds)	Native	Singularity
Mean	5153.4	5966.4
Standard Deviation	9.6290	11.287
Minimum Value	5137.0	5940.0
Maximum Value	5175.0	5983.0

From the mean runtime, we can see that there is a clear difference between the native and Singularity deployment models. In fact, the mean Singularity runtime is nearly 16% greater than the native runtime. The two-tailed t-test for this benchmark yields a critical t-value of 2.01 and again we use a threshold p-value of 0.05. Running the t-test yields a t-statistic of -284.73 and a p-value of 1.048×10^{-84}. The absolute value of our t-statistic is greater than 2.01 and the resulting p-value is far below 0.05 so we can reject H_0 for this benchmark, i.e. there is a statistically significant difference in the runtime between native and Singularity.

Translation - Non Recurrent (Translation) The Translation benchmark is an intensive benchmark performance-wise, so we had to reduce the target BLEU score from 25 to 20 in order to get it to converge in a reasonable time for us to measure its results (The HPC jobs were limited to 96 hours of runtime and translation did not converge with a BLEU score of 25 in the time limit). With these updates, we were able to complete 28 runs of translation on the Type-1 (HPC node). As we can see in figure 7, the runtimes are almost identical. The outliers for the Singularity and Native are caused by a single experiment that did not converge but instead completed all training epochs. This gives an indication of the maximum expected training time for translation when using the default of 10 training epochs.

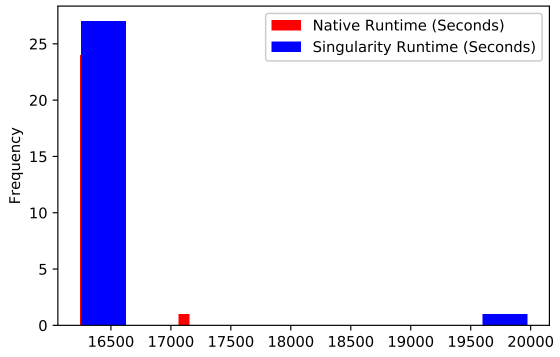

Figure 7: Histogram of runtime in seconds of Translation benchmark run on Type-1 (HPC node)

Table 7 shows the statistics for the run and the results confirm what we see in figure 7. The statistics for both the native and Singularity runs are remarkably similar.

Table 7: Translation Type-1 (HPC node) Results

Statistic (seconds)	Native	Singularity
Mean	16327.57	16439.9
Standard Deviation	174.12	697.39
Minimum Value	16244.0	16251.0
Maximum Value	17156.0	19970.0

We ran the two-tailed independent t-test on this benchmark with our standard parameters, critical t-value: 2.01, p-value: 0.05. The results of the test yield a critical t-value of -0.82712 and a p-value of 0.41180. Thus, since p-value > 0.05 and the absolute value of the t-statistic < 2.01, we fail to reject the null hypothesis for this benchmark.

Recommendation A histogram of the recommendation benchmark results for the Type-1 HPC is displayed in figure 8. In table 8 we see that the mean runtime for recommendation in the Singularity container is actually lower than the mean runtime for recommendation in native. To determine the significance of this, we ran the two-tailed independent t-test with the standard parameters (2.01, 0.05). The results of this gave us a t-value of 1.87 and a pvalue=0.0654. These values are close to the thresholds that we have set, but not are not quite enough for us to reject the null hypothesis since p-value > 0.05 and $|t| < 2.01$. Thus, we fail to reject H_0 for the recommendation benchmark.

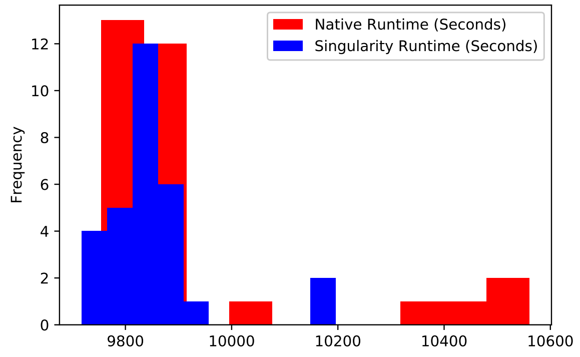

Figure 8: Histogram of runtime in seconds of Recommendation benchmark (Type-1)

A histogram of the results for Type-2 (HPC node) is shown in figure 9. Here we see a similar outcome to Type-1 (HPC node). Again, it appears that the Singularity runtime is actually lower than the mean runtime as a native process.

Table 9 confirms what we see in figure 9. The mean runtime for recommendation in the Singularity container is actually slightly lower than the native runtime. Due ongoing system maintenance with the Type-2 (HPC node), we executed only 20 runs of recommendation and randomly sampled 15 for the results displayed in the histogram. The 15 results for each type of run yields a critical t-value of 2.048.

The results of the two-tailed independent t-test yields a t-value of 3.6489 and a p-value of 0.001. As the t-value is above the critical t-value and the p-value is below our threshold we have an interesting scenario for rejecting the null hypothesis for this benchmark.

Table 8: Recommendation Type-1 (HPC node) Results

Statistic (seconds)	Native	Singularity
Mean	9932.03	9849.80
Standard Deviation	217.06	102.07
Minimum Value	9754.00	9718.0
Maximum Value	10560.0	10196.0

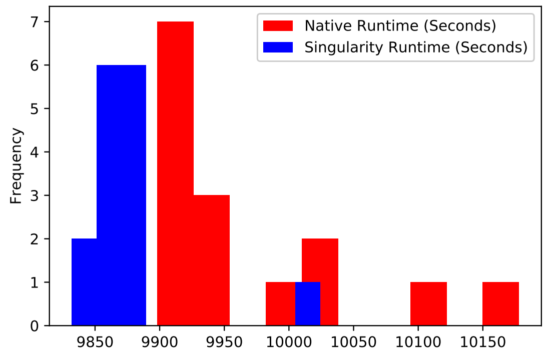

Figure 9: Histogram of runtime in seconds of Recommendation benchmark run on Type-2 (HPC node)

Table 9: Recommendation Type-2 (HPC node) Results

Statistic (seconds)	Native	Singularity
Mean	9966.7	9877.9
Standard Deviation	83.904	42.939
Minimum Value	9898.0	9832.0
Maximum Value	10178.0	10024.0

6 CONCLUSIONS

This paper explored the use of ARC containers for AI workloads. We used the emerging MLPerf benchmark to characterize performance overhead of Singularity containers. We ran experiments with selected benchmark applications deployed as native processes and deployed as Singularity containers. The overall results of the hypothesis testing shows us the following:

- Sentiment Analysis - No statistically significant difference between native process and Singularity runtimes
- RNN Translator - Statistically significant overhead for Singularity vs. native process with multiple GPUs. No statistically significant difference for single GPU
- Translation - No statistically significant difference between native process and Singularity runtimes
- Recommendation - No statistically significant difference on Type-1 (HPC node) but a slightly statistically significant overhead for native process over Singularity

6.1 Limitations

Our experimental design has some obvious limitations. We collected a relatively small number of samples for the experiments. Additionally, the compatible configurations are similar but not identical. Differences in the runtime environments could contribute to variance in the experimental results. Also, we did not evaluate multi-node distributed learning.

6.2 Future Work

Some potential future work includes evaluating additional configurations such as distributed learning (multi-node/single-GPU, multi-node/multi-GPU) for overhead as these configurations were not achievable for us. With MLPerf, some future improvements include containerizing data download and preparation and separating download, validation, and preparation into distinct steps. Each step should produce data which should be immutable. Additionally, ensuring that the data location is independent of the underlying framework so it can be shared between benchmark implementations in other frameworks will be helpful for future research.

In conclusion, overall, it seems that with a single GPU, there is no statistically significant difference between the runtime in native process and in a Singularity container. However, as we notice with RNN Translator, when utilizing multiple GPUs for the workload, there is a measurable overhead for Singularity versus a native process. The lack of overhead for single GPU workloads however, is good news for portability and reproducibility of results. If a Singularity container can be built and shared then that eliminates the hassle of having to build the environment required for the workload on different machines.

ACKNOWLEDGMENTS

The authors acknowledge Department of Defense HPC Modernization program for providing computational resources and technical support that have contributed to the results reported within this paper. URL: https://www.hpc.mil/.

REFERENCES

[1] [n. d.]. Google Cloud including GCP & G Suite âĂŤ Try Free. ([n. d.]). https://cloud.google.com/
[2] [n. d.]. Reference implementations of training benchmarks. ([n. d.]). https://github.com/mlperf/training original-date: 2018-03-29T21:56:06Z.
[3] [n. d.]. What is DevOps? - Amazon Web Services (AWS). ([n. d.]). https://aws.amazon.com/devops/what-is-devops/
[4] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation*. Association for Computational Linguistics, Berlin, Germany, 131–198. http://www.aclweb.org/anthology/W/W16/W16-2301
[5] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris RÃľ, and Matei Zaharia. [n. d.]. Analysis of the Time-To-Accuracy Metric and Entries in the DAWNBench Deep Learning Benchmark. ([n. d.]), 9.
[6] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 171–172.
[7] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.
[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173–182. https://doi.org/10.1145/3038912.3052569
[9] Joshua Higgins, Violeta Holmes, and Colin Venters. [n. d.]. Orchestrating Docker Containers in the HPC Environment. In *High Performance Computing* (2015) *(Lecture Notes in Computer Science)*, Julian M. Kunkel and Thomas Ludwig (Eds.). Springer International Publishing, 506–513.
[10] Rie Johnson and Tong Zhang. 2014. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *CoRR* abs/1412.1058 (2014). arXiv:1412.1058 http://arxiv.org/abs/1412.1058
[11] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. [n. d.]. Singularity: Scientific containers for mobility of compute. 12, 5 ([n. d.]), e0177459. https://doi.org/10.1371/journal.pone.0177459
[12] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. http://www.aclweb.org/anthology/P11-1015
[13] NVidia. [n. d.]. nvidia/cuda. ([n. d.]). https://hub.docker.com/r/nvidia/cuda
[14] Reid Priedhorsky and Tim Randles. [n. d.]. Charliecloud: Unprivileged Containers for User-defined Software Stacks in HPC. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017) *(SC '17)*. ACM, 36:1–36:10. https://doi.org/10.1145/3126908.3126925 event-place: Denver, Colorado.
[15] Cristian Ruiz, Emmanuel Jeanvoine, and Lucas Nussbaum. [n. d.]. Performance Evaluation of Containers for HPC. In *Euro-Par 2015: Parallel Processing Workshops* (2015) *(Lecture Notes in Computer Science)*, Sascha Hunold, Alexandru Costan, Domingo GimÃľnez, Alexandru Iosup, Laura Ricci, MarÃŋa Engracia GÃşmez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander (Eds.). Springer International Publishing, 813–824.
[16] Netflix Technology. [n. d.]. How We Build Code at Netflix. ([n. d.]). https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15
[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. (2017). arXiv:cs.CL/1706.03762
[18] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ÅĄukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. (2016). arXiv:cs.CL/1609.08144
[19] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose. 2013. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 233–240. https://doi.org/10.1109/PDP.2013.41
[20] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K. Panda. [n. d.]. Is Singularity-based Container Technology Ready for Running MPI Applications on HPC Clouds?. In *Proceedings of the10th International Conference on Utility and Cloud Computing* (2017) *(UCC '17)*. ACM, 151–160. https://doi.org/10.1145/3147213.3147231 event-place: Austin, Texas, USA.