Multi-Modal Multi Environment Performance Comparison on Google Cloud Platform

Aman Chopra*
Computer Science Department
New York University
New York, NY, USA
ac8511@nyu.edu

Nidhi Ranjan*
Computer Science Department
New York University
New York, NY, USA
nr2387@nyu.edu

Shiv Ratan Sinha*
Computer Science Department
New York University
New York, NY, USA
srs9969@nyu.edu

***Abstract*—In recent years, due to the enormous amount of data, and the advancements in Big Data and Machine Learning Applications, Cloud Computing has become ubiquitous. Most of the applications in today's world are deployed to the cloud so that they can easily be scaled and accessed by the end-users. Along with deploying the applications in the bare Virtual Machines, the advent of containerized Software has given the developers a hard time to choose the environment where they should deploy their applications. There are different kinds of applications and Machine Learning workloads and it becomes very important to choose the best environment where the application should be deployed for maximum resource utilization.**

In this paper we have tried to compare three environments i.e., the bare Virtual Machine, Docker running inside the Virtual Machine, and Singularity running inside the Virtual Machine on two different workloads i.e., an MNIST digit classifier using Convolutional Neural Network and a Sentiment Analyzer using Recurrent Neural Network. We have used the same underlying hardware and have run all these workloads using a GPU-enabled Virtual Machine inside the Google Cloud Platform to better compare the metrics and understand the implication of the change of metrics in different environments on the performance of the application.

Index Terms*—MNIST, Sentiment Analysis, CNN, RNN, Virtual Machine, Docker, Singularity, nsys, nvprof, Roofline Model, CUDA, Graphics Processing Unit, Google Cloud Platform

*All the authors contributed equally to this work.

I. Introduction

In today's world, companies and developers have a plethora of options to choose from in order to deploy their applications. First, there are so many cloud services to choose from. After that, there are also a lot of container platforms available. This drastically increases the number of options. There is of course no clear winner as the usefulness of one service or environment totally depends on the problem statement, but it is also important to understand the factors that affect the performance of the application in order to choose the right platform and environment for maximum resource utilization.

For our research, we have used the Google Cloud Platform as the Cloud Service Provider. The monthly uptime percentage of GCP is $\geq$ 99.9% for instances in multiple zones. GCP also offers 24 regions with 73 total zones. According to [1], the Google Cloud Platform also supports the maximum number of databases & has a great repository of in-built libraries like CUDA. The above reasons along with the 300$ worth of free credits the new user gets compelled us to use GCP as the platform to perform our comparative study.

After choosing the platform, we had to choose the environments. We wanted to compare the workloads on the bare Virtual Machine and the containers. As we know that Docker and Singularity are two of the most widely used containers so we chose those two for our study. We wanted to take two different kinds of workloads using a different underlying architecture to increase the scope of our observations. We took a baseline model of the MNIST Digit Classifier using a Convolutional Neural Network designed using pytorch. For the second model, we took a baseline model of Sentiment analyzer using Recurrent Neural Network also designed using pytorch. We did all our analysis on these two models on GCP and compared the metrics on the three environments to better understand how the performance is impacted by the environments and what kind of environment is suitable for which kind of workload.

The paper has been laid out in the following manner. Section II, the objective section deals with the primary objectives and the problem description of this paper. Section III of this paper talks about the key studies being already done in this field. Section IV lays down the proposed idea and methodology including the setup we have followed to compare the three environments i.e., bare Virtual Machine, Docker running inside the VM, and Singularity running inside the VM on two classification workloads i.e., a baseline MNIST digit classifier using Convolutional Neural Networks and a baseline Sentiment Analyser using Recurrent Neural Networks. Section V discusses the experiments and Section VI the results. Section VII talks about the learnings and the challenges, and Sections VIII concludes this paper.

II. Objective

In this paper, the authors want to compare different models on different environments like Virtual Machine, Docker, and Singularity in Google Cloud Platform. The authors want to gain insight on how factors like memory, FLOPS, communication overhead, throughput, etc, on different platforms, affect the performance of the application. The results are obtained by diving deep into the GPU metrics of different workloads in different environments. This research can help understand what kinds of workloads utilize resources in the best possible way in which environments when all the environments are using the same underlying hardware and the GPU.

III. Literature Survey

A lot of studies have been done to compare the performance of different applications on different containers. In [2] the authors have trained two networks LCNN and SNN, on the MNIST dataset, and compared the performance of Docker, Singularity, and CharlieCloud. The authors have based their observation just on the time the environment took to perform the operations in the workload. According to this study, Singularity should be the chosen choice among the other containers. In [3] the authors utilized the MLPerf benchmark to evaluate the performance difference between two application deployment models: process-based and container-based. They analyzed the Singularity container performance overhead for complex Deep Learning training and inference tasks. In [4] the authors compared the prediction and the accuracy of models trained on Docker, Singularity, and Open Stack. According to them, Docker had the best performance with the lowest execution time in all experiments done with all the technologies. Most of the research that we have gone through has based their observations either on execution times or test/training accuracies. We believe that there is a lot that goes inside it. We also believe that a detailed investigation of the GPU metrics including the memory, and communication overhead is needed to understand why the difference in time is observed. This will give readers more insight into how different metrics in different workloads affect their performance

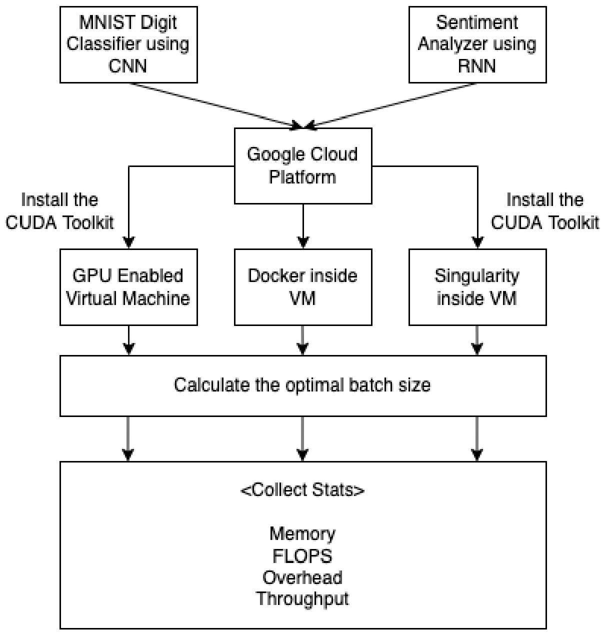

Figure 1: Methodology

in different environments. This would also help the readers make an informed decision as to what environments can be used for which kind of workloads.

IV. Proposed Idea and Methodology

Figure 1 shows the high-level overview of the methodology we have followed. We have first chosen two different workloads. We fed both the workloads into the Google Cloud Platform where we provisioned a GPU-enabled Virtual Machine. We installed the CUDA toolkit and Singularity to the Virtual Machine. Note that the Docker Container was already installed in the Virtual Machine. After that, we calculated the optimal batch size for both workloads for maximum GPU utilization. Once we have all these things calculated, we performed the experiments calculating and comparing different GPU metrics in different environments.

A. *Workloads*

We have chosen the workloads of varying underlying architecture written using the same Machine Learning Framework to better compare the performance on different platforms. The two workloads we chose are as follows:

*1) **MNIST Image Classifier**:* This model helps in classifying the MNIST dataset's digit into 10 categories (0-9).

Model details: This CNN model uses 2 convolution layers, 2 dropouts, and 2 fully connected layers. The loss function that is being used is the negative log likelihood loss.

Dataset: We have used the MNIST dataset for digit classification. The original MNIST dataset has around 60,000 images for training and 10,000 images for testing. We have used a subset of this dataset and have used 512 images for training our model.

*2) **Sentiment Analyzer**:* This model helps in binary classification of the sentiment of a sentence as either positive or negative.

Model details: This RNN model uses 1 Embedding layer, 1 RNN layer, 1 Fully Connected layer The loss function that is being used is the Binary Cross Entropy loss as it is a binary classification problem.

Dataset: We have used the IMDB dataset to identify the sentiment of the movie review as positive or negative. The original IMDB dataset has around 50,000 reviews out of which 25,000 are used for training, 10,000 are used for testing, and 15,000 are used for validation. We have used a subset of this dataset and have used 1,000 images for training our model.

B. *GCP Virtual Machine*

We provisioned a GPU Enabled Virtual Machine. We used a Debian 10 based Deep Learning VM image with CUDA 11.3. This was an extremely good feature provided by GCP as it easily installed all the drivers needed to run the GPU along with the CUDA Toolkit. The details of the Virtual Machine are listed in Table I

Here are the steps to provision a GPU Enabled Virtual Machine from the Google Cloud Platform:

1) First we need a paid account as we cannot use the free credits to provision a GPU.
2) Before we provision a GPU, we need to go to the section ”All quota” in GCP.
3) We need to elect ”gpu” in the filters to see all the regions where we can get a GPU from. Different regions have different types of GPUs, so we need to decide which GPU and accordingly the region which would be suitable based on our location.
4) We need to increase the Global GPU quota to 1 and give a reason as to why we need a GPU.
5) We also need to increase the specific region GPU Quota to 1 and give a reason as to why we need a GPU.
6) Both these requests are sent to the Google Cloud Team, and based on the reasoning they approve the request. This process usually takes around 10 minutes.
7) We can then provision a Virtual Machine and use it. The rates are based on per-hour usage.
8) Using the Debian Deep Learning on Linux Image, we can directly run the ”install.sh” scripts which install all the required drivers and CUDA Toolkit so we can conveniently use the GPU. Without this image, we encountered several issues with version mismatches and driver installations.

C. *Environments*

Our first comparison is between a Virtual Machine and Container. After which, we want to compare the performance and metrics on different containers as well like Docker and Singularity. This makes the total number of environments 3 for our study. Before performing any experiments, we wanted to know in-depth about the philosophy of the environments we would be using. Table II shows the key differences between a Virtual Machine and a Container. Table III shows the key differences between Docker and Singularity as inspired from [5].

Table I: Virtual Machine specifications

Machine Configuration	
Specification	**Value**
Operating System	Debian Deep Learning on Linux
Version	Debian 10 based DL VM with CUDA 11.3
Boot disk type	Balanced persistent disk
Size	100GB
GPU type	NVIDIA Tesla V100
Number of GPUs	1
Machine type	n1-standard-1
vCPU	8@2.3GHz
RAM	30 GB

Table II: VM vs Containers

VM	**Containers**
Hardware-level process isolation	OS-level process isolation
Spin-up time can take minutes	Spin-up time can take seconds
VMs can move to new host easily	Containers are destroyed and re-created rather than moving

Table III: Docker vs Singularity

Docker	**Singularity**
Binds and mounts Host File System to access data	Singularity is an executable that runs as a user process
No in-built support for OpenMPI	Built-in support for OpenMPI
Starts with privileged access	Starts with normal user access. su or sudo to become root user cannot be used
Utilize Networks Namespace where Network Address Translation is the default	Transparent. Access network like any user process would

*1) **Bare Virtual Machine:*** It is the first environment where we perform our experiments.

This is the environment we get as soon as we provisioned our Virtual Machine. We pull our code using GitHub and run the experiments directly on the GPU-enabled Virtual Machine.

*2) **Docker**:* Docker came preinstalled with Debian Deep Learning on Linux. We just had to write our dockerfile, build an image using it, and run the container using the image. Source Code 1 is a dockerfile we wrote to make a docker image for our CNN-based model for digit classification. We used the NVIDIA container runtime provided by the NVIDIA Container Toolkit to install drivers and use the profilers inside Docker.

Source Code 1: dockerfile for docker

```
FROM pytorch/pytorch
RUN apt-get update
USER root
COPY main.py .
COPY nvprof.sh .
COPY nsys.sh .
```

*3) **Singularity**:* We had to manually install Singularity for our Virtual Machine as it was not already installed. Here are the steps we followed to install Singularity.

1) We first had to update apt and install build essentials like libseccomp-dev pkg-config squashfs-tools cryptsetup.
2) We then had to download the Go language as it is required by Singularity and set the required PATH variables.
3) We then had to clone the Singularity code from GitHub and then had to run the mconfig. After this, we had to " make install". We used the command "singularity version" to check if the Singularity has been installed properly in the machine.

Source Code 2 is a dockerfile we wrote to make a singularity image for our CNN-based model for digit classification.

Source Code 2: dockerfile for singularity

```
FROM pytorch/pytorch
RUN apt-get update
USER root
RUN apt-get install -y vim
RUN pip3 install bc
```

We had to push this file to dockerhub and pull it to make a sif file. We created a sif image from the custom docker image and ran this as sudo because of the profiling permission issues inside Singularity. We also used the nv flag to use the GPU inside Singularity. The command we used: sudo singularity shell –bind /usr/local/cuda –nv mnist.sif bash

*D. **Tools***

The main tool we have used for GPU metrics calculation is nvprof [6]. This is part of the CUDA Toolkit to analyze the programs running on the GPU. This can tell us about the communication overhead, the time taken for allocating memory and copying data, the kernel efficiency, etc. Nvprof also gives a lot of metrics and events like *warp_execution_efficiency, dram_read_throughput, dram_read_transactions, flop_count_sp,* etc. Using cuda_profiler_api.h header file, we can profile a specific code segment as well. This is helpful in just profiling the kernel using cudaProfilerStart() and cudaProfilerStop() functions. We have also used the NVIDIA Nsight Systems tool[7] to calculate the kernel execution time. The Nsight Systems CLI provides a simple interface to collect on a target without using the GUI.

*E. **Language Constructs and functions***

For CUDA, these are the important terms and functions:

Table IV: The Setup

Platform/Framework/Language	Version
Docker	20.10.4
Singularity	3.6.3
CUDA Toolkit	11.3
Python	3.7

1) **Kernel**: This is a function executed in the GPU. Every CUDA kernel starts with a __global__ declaration specifier. Programmers provide a unique global ID to each thread by using built-in variables.
2) **cudaMemcpy**: This copies the memory pointed by the source to the destination. This can either be from the host to the device or from the device to the host.
3) **cudaMalloc**: This function allocates bytes of memory in the device and returns a pointer. After the work is done, it is the responsibility of the programmer to free the device memory is cudaFree().

F. *Setup*

In this section, we would be talking about the versions of the platforms, libraries, and frameworks we have used along with the GPU Configurations. Table IV details the frameworks, languages, and platforms we have used along with the versions. Table V contains the GPU Configuration. We have used NVIDIA Tesla V100 GPU. The main specifications to notice here are Tensor Performance and Memory Bandwidth. Since the Tensor FLOPS are enabled by default in pytorch, we have used Tensor Performance for the Roofline analysis. The Memory Bandwidth we would be using for our Roofline analysis is 900GB/sec.

V. Experiments

A. *Batch Calculation*

Before doing any calculation, we thought it would be important to empirically calculate the optimal batch size for the maximum GPU utilization. In order to do so, we calculated the real-time using the time Linux time command to identify which batch size among the chosen ones was taking the least amount of time. This helped us perform the experiments on the batch size which was suitable for the chosen problem statement and hardware. Figure 2 shows the real-time for different batch sizes for the MNIST Digit Classification. The least amount of time was taken for batch size 64. Similarly, Figure 3 shows the real-time for different batch sizes for the Sentiment Analyzer. The least amount of time was taken for batch size 128. All the experiments are performed for 1 epoch.

Table V: GPU specifications

GPU Configuration	
Specification	**Value**
Model	NVIDIA Tesla V100 SXM2
NVIDIA Tensor Cores	640
NVIDIA CUDA Cores	5,120
Tensor Performance	125 TFLOPs
GPU Memory	32GB HBM2
Memory Bandwidth	900GB/sec
System Interface	NVIDIA NVLink
Max Power Consumption	300W

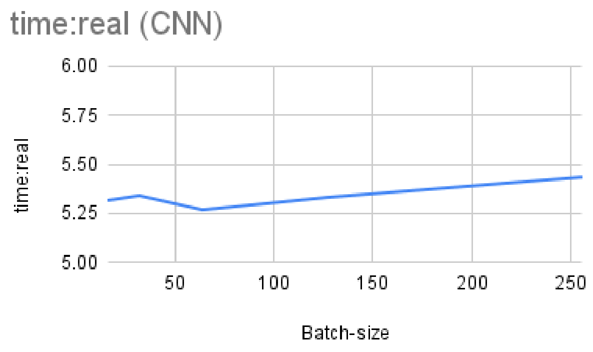

Figure 2: Optimal batch size of CNN

B. *Roofline Analysis*

It is often misunderstood that the performance of a program depends solely on the machine. It is far from true. How the program is

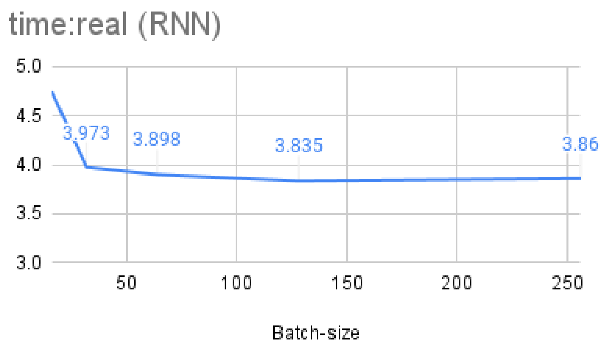

Figure 3: Optimal batch size of RNN

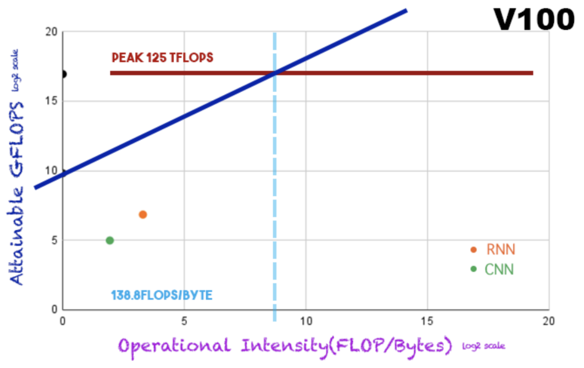

Figure 4: Roofline Analysis of both the models

written, and how compute and memory friendly the program is can significantly improve the performance of the program. Of course, there is an upper limit on the computation and the memory bandwidth, and this limit is set by hardware but within that limit, there is always scope for improving the program. This is where the roofline modeling comes into the picture. It combines the software and the hardware characteristics to show where the performance of the program lies and whether the program is memory-bound or compute-bound. It can provide us important insights into how the performance of the application can be improved.

We performed the roofline analysis for both the models in the bare Virtual Machine. We calculated the Arithmetic Intensity of both the models by calculating the total number of FLOPS and the memory transactions. To calculate the FLOPS/sec, we used the nsys tool in order to calculate the time. To calculate the upper bound determined by the hardware, we used the specifications provided by the NVIDIA Documentation for NVIDIA Tesla V100. The peak Memory Bandwidth was 900 GB/sec and the peak TFLOPs was 125. Figure 4 shows the roofline analysis of both the models. Both axes are in logarithmic scale as we have taken the base as 2. We can see that both the models are memory-bound. We cannot compare the performance of 2 models as both are handling different workloads, but it can be said that the RNN model performs better in terms of memory utilization.

C. *CUDA memset*

Memset is required to allocate the data before performing an operation on the GPU cores. Figure 5 and Figure 6 show the time it took for memset in all the environments in the CNN and RNN based models respectively. We can see that the CUDA memset operation took slightly more time in the Docker in both the models. This is because of the fact that Docker runs in the NVIDIA container runtime provided by the NVIDIA Container Toolkit. The NVIDIA Container Toolkit is a collection of packages which wrap container runtimes with an interface to the NVIDIA driver on the host. It focuses more on running containers and setting up namespaces and cgroups for containers. This adds an extra layer in the case of Docker because of which it takes more time for the CUDA memset. Singularity, however, is like an executable, and hence the communication overhead in the case of a bare Virtual machine and Singularity is lesser compared to Docker.

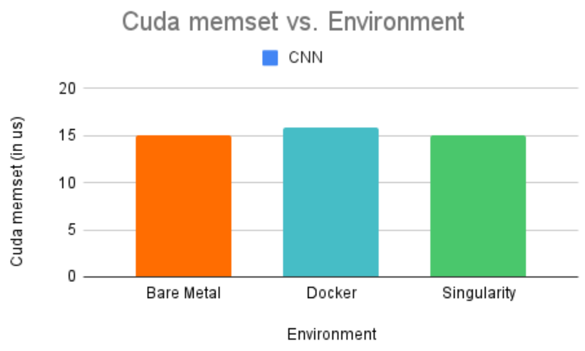

Figure 5: Memset vs Environment for CNN

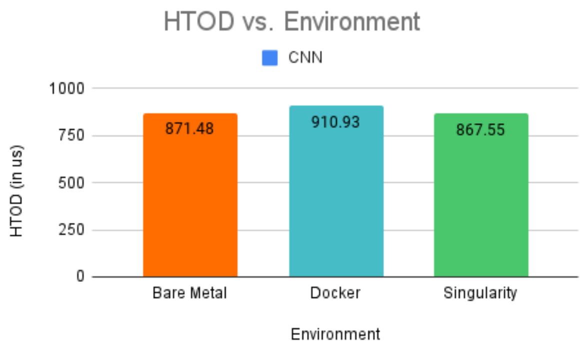

Figure 7: HTOD vs Environment for CNN

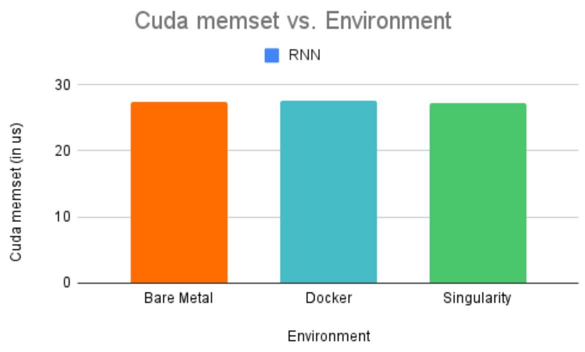

Figure 6: Memset vs Environment for RNN

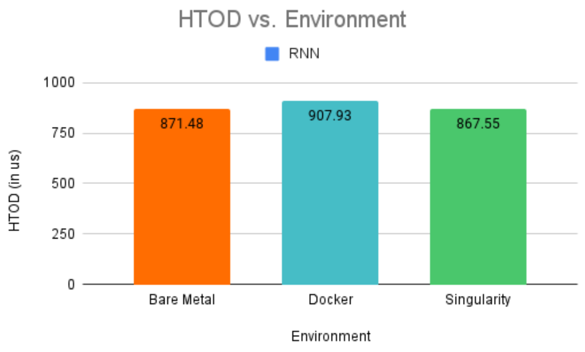

Figure 8: HTOD vs Environment for RNN

D. *CUDA memcpy HTOD*

HTOD represents Host to Device. Memcpy is required to copy the data from the host to the device (GPU) so that the GPU can perform the operations. Since GPU is a peripheral device, data needs to be transferred to the GPU so that it can perform the computations. Figure 7 and Figure 8 show the time it took for copying data from the host to the device in all the environments in the CNN and RNN based models respectively. We can see that the CUDA memcpy operation took slightly more time in the Docker in both the models. This is because of the similar reason we had for CUDA memset. Overall, we see that the communication overhead with Docker is slightly more than that of other environments.

E. *Throughput*

We calculated both the read and write throughputs for all the environments in both the models. According to our experiments, the average of read and write throughput is independent of the environment as it was the same in all the environments for a particular model.

*1) **DRAM Read Throughput:*** DRAM is the physical device memory. DRAM is accessed on L2 misses. DRAM Read Throughput denotes the number of read operations in DRAM per second. One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second, for items up to 4 KB in size. Figure 9 and Figure 10 show the DRAM Read Throughput of all the environments in the CNN and RNN-based

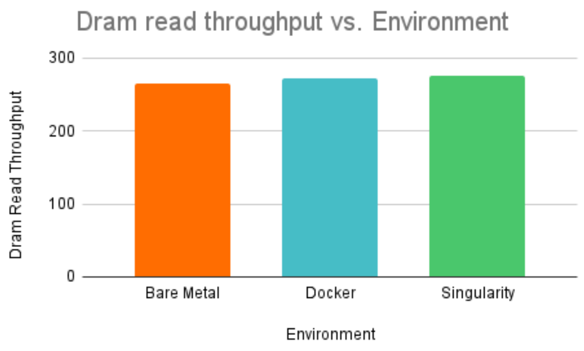

Figure 9: DRAM Read Throughput (CNN)

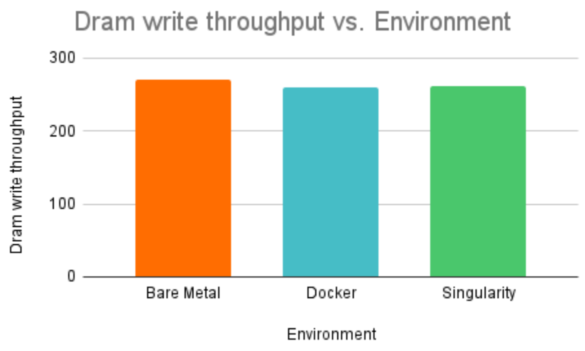

Figure 11: DRAM Write Throughput (CNN)

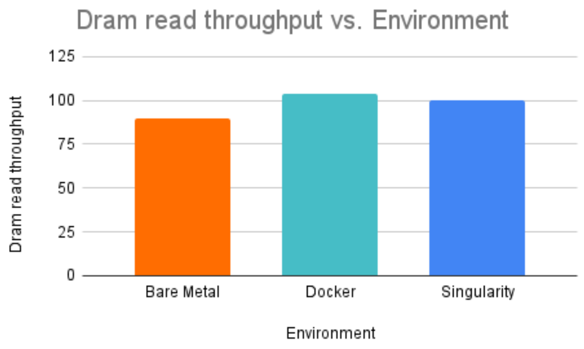

Figure 10: DRAM Read Throughput (RNN)

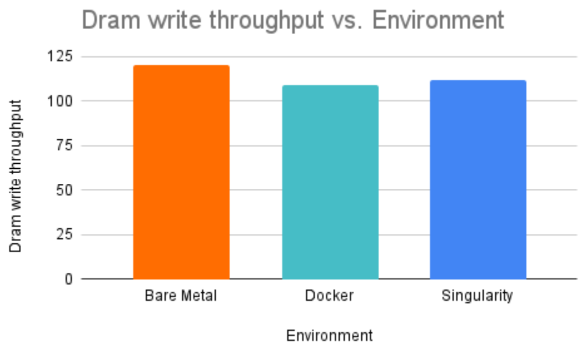

Figure 12: DRAM Write Throughput (RNN)

models respectively.

*2) **DRAM Write Throughput:*** It denotes the number of write operations per second. One write capacity unit represents one write per second for items up to 1 KB in size. Figure 11 and Figure 12 show the DRAM Write Throughput of all the environments in the CNN and RNN-based models respectively.

*F. **FLOPS***

Figure 13 and Figure 14 show the FLOPS of all the environments in the CNN and RNN-based models respectively. It was very surprising to see that the FLOPS of the bare Virtual Machine was way less than that of Docker and Singularity. FLOPS is usually a model property and it should not depend on the environment. On digging deeper, we found that the kernels being used were different for different environments and this behavior was the same across multiple runs. We have attached a ”kernels_vm_docker_singularity_64.txt” file which shows that different kernels were used for the bare Virtual Machine and the Containers. According to our research, pytorch and other DL frameworks use NVIDIA CUDA Deep Neural Network library (cuDNN) primitives to leverage the kernels. This indicates that a change in kernel might be attributed to a change in the version of CUDA libraries. In our case, however, the binaries of the CUDA libraries were the same across the 3 environments and yet we saw this different behavior.

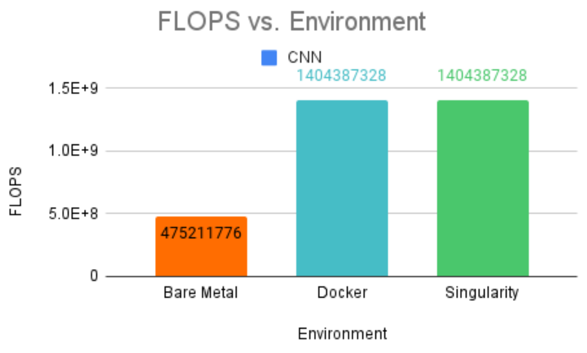

Figure 13: FLOPS vs Environment for CNN

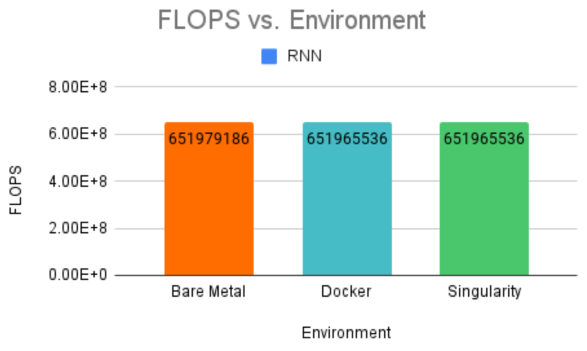

Figure 14: FLOPS vs Environment for RNN

G. *HBM Transactions*

HBM stands for high bandwidth memory and is a type of memory interface used in 3D-stacked DRAM. Figure 15 and Figure 16 show the HBM Transactions of all the environments in the CNN and RNN-based models respectively. We can see that the HBM Transaction in the case of the bare Virtual Machine for CNN is way higher. This difference in values can be attributed to the usage of different kernel sets. In GPU there are two kinds of operations when computing the value of a layer in a deep neural network. The Multiply and accumulate (MAC) operation takes three inputs i.e., the kernel, the input data, and the intermediate calculated data. The inputs are transferred through the Arithmetic Logic Unit and the outputs are stored in

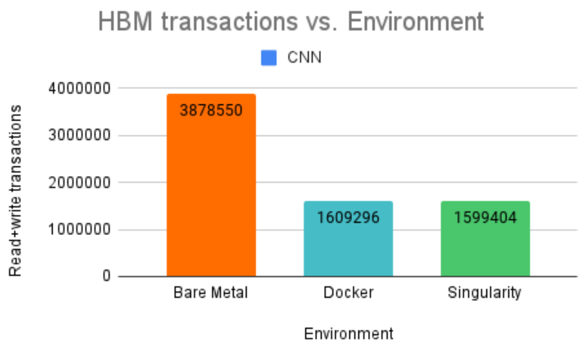

Figure 15: HBM Transactions vs Environment for CNN

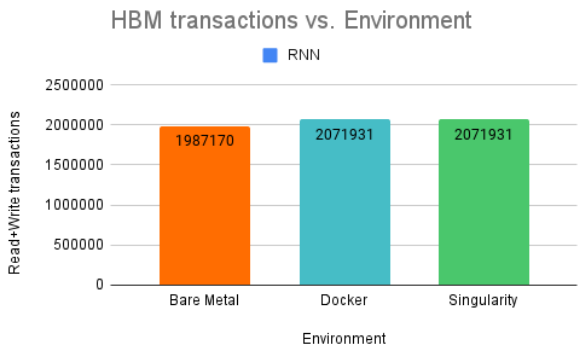

Figure 16: HBM Transactions vs Environment for RNN

a separate memory unit. The Memory Access Operation stores data in a designated memory unit, depending on the size of the data. The size and time consumption of the memory unit increases with the size of the data. In the worst case, reading and writing to the DRAM can be very costly.

Due to a difference in kernel sets, the size of data being stored varies which eventually varies the type of the memory being used like the registers, buffer, or DRAM. This is the reason for the apparent difference in HBM transactions for different environments in the case of CNN.

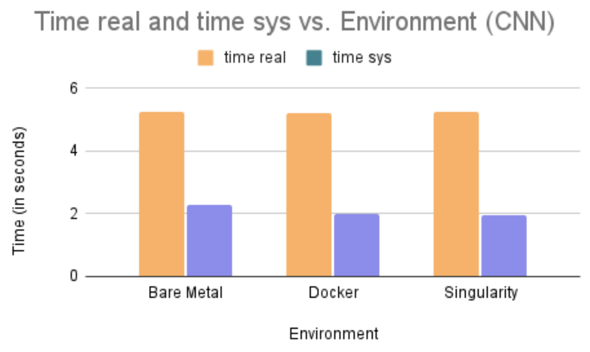

Figure 17: Real and System Time vs Environment for CNN

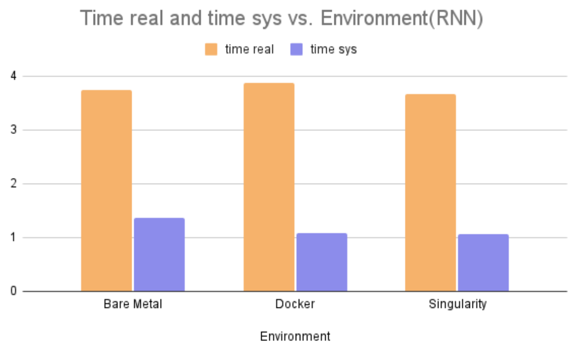

Figure 18: Real and System Time vs Environment for RNN

H. *Time*

Figure 17 and Figure 18 show the Real and the System Times of all the environments in the CNN and RNN-based models respectively. We can see that both the Real and System times are more or less the same in all the environments for a particular workload. Even though the number of FLOPS was way lesser in the case of bare Virtual Machine, the similar time can be attributed to the more number of High Memory Bandwidth Transactions in the case of bare Virtual Machine. We can also see that the time taken by Singularity is relatively less compared to the other environments. This is because of the fact that Singularity has native support for Graphics Processing Units.

VI. Results

The key results we obtained from this research are:

1) The memory overhead of Docker was relatively more than bare Virtual Machine and Singularity.
2) The average of the read and the write throughput was the same for all the environments in a particular model.
3) The FLOPS in the case of bare Virtual Machine for MNIST Digit Classification was way lesser because of the usage of different kernel sets.
4) The High Memory Bandwidth Transactions of the bare Virtual Machine were way more than the other environments for MNIST Digit Classification. This is again because of the usage of different kernel sets.
5) The time taken by all the environments for 1 epoch for a particular workload was pretty much the same. Singularity is preferred over other containerized environments because of the native support for GPU and the security it offers.

VII. Learnings and Challenges

A. *Learnings*

The key insights we got via this research are:

1) The FLOPs for the same model can differ in different environments. We have seen this in the case of MNIST Digit Classification using Convolutional Neural Network.
2) FLOPs cannot be used as a universal proxy for efficiency, it depends on which type of Memory and Accumulate operations are being done [8]. As we have seen in the case of FLOPS and HBM Transactions, even though the FLOPS were way

lesser in the case of bare Virtual Machine for CNN, the time taken by the model for the same number of epochs was the same for all the environments.

3) Singularity has native support for the GPU and hence its performance was slightly better when compared with Docker.
4) Both the workloads performed similarly in the 3 environments, but we believe for the containerization purposes, singularity wins. This is because it is relatively more secure, has native GPU support, and can be used with Kubernetes as well.

B. Challenges

There were a few challenges we faced while conducting this research. Those were:

1) We could not use the free credits to get the GPUs in Google Cloud Platform. An upgraded account was necessary. We also needed to send a request citing the reasons needed for the GPU to the Google Cloud Platform team.
2) We encountered some CUDA Toolkit version and GPU driver installation issues both in the case of the bare Virtual Machine and Docker.
3) We also encountered issues installing Singularity inside the Virtual Machine.
4) We encountered permission issues with the GPU Profiling, especially in the case of Singularity.
5) ncu support is removed from cheaper GPU options like NVIDIA Tesla P4, so we had to resort to the costlier options.

VIII. Conclusion and future work

We have analyzed and compared two different workloads CNN and RNN in three different environments. We have observed some interesting results in the case of FLOPS and the High Bandwidth Transaction calculation. This research can be helpful in gaining a deep insight into the GPU metrics in different environments. Apart from the time, we also compared the memory overhead, throughput, FLOPS, and transactions to gain a deeper understanding of the results we got.

In the future, we want to explore more about why and how a particular model chooses a kernel. We want to profile the State of the art models instead of profiling the baseline models. We also want to profile more containers like Charlie cloud and compare its performance with other containers like Docker. There are some researches that show the performance of Charlie Cloud is better than Docker. We want to compare Charlie Cloud with Singularity for the State of the art models. We also want to experiment with unikernels and profile other models like LSTM and Transformers to see if our results hold with other models as well.

References

[1] P. Shelke, M. Talati, and R. Chinchamalatpure, “Comparative study of cloud platforms -microsoft azure, google cloud platform and amazon ec2,” *Journal of Research in Engineering and Applied Sciences*, vol. 05, pp. 60–64, 04 2020.

[2] M. Sun, Y. Guo, hu Zhang, W. Cao, and M. Yuan, “Performance comparison of multiple containers running artificial intelligence applications,” *Journal of Physics: Conference Series*, vol. 1948, no. 1, p. 012005, jun 2021. [Online]. Available: https://doi.org/10.1088/1742-6596/1948/1/012005

[3] M. Newlin, K. Smathers, and M. DeYoung, “Arc containers for ai workloads: Singularity performance overhead,” 07 2019, pp. 1–8.

[4] J. Marquez and M. Castillo, *Performance Comparison: Virtual Machines and Containers Running Artificial Intelligence Applications*, 01 2021, pp. 199–209.

[5] Docker vs singularity vs shifter vs uge container edition. [Online]. Available: https://tin6150.github.io/psg/blogger_container_hpc.html

[6] Cuda toolkit documentation. [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html

[7] Nvidia developer tools documentation. [Online]. Available: https://docs.nvidia.com/nsight-systems/UserGuide/index.html

[8] Are all flops created equal? [Online]. Available: https://towardsdatascience.com/are-all-flops-created-equal-a-comparison-of-flops-vs-run-time-3cebb2dcb3da