Comparison Between Bare-metal, Container and VM using Tensorflow Image Classification Benchmarks for Deep Learning Cloud Platform

Chan-Yi Lin, Hsin-Yu Pai and Jerry Chou
Computer Science, National Tsing Hua University, No.101, Sec. 2, Guangfu Rd., East Dist., 300, Hsinchu, Taiwan

Keywords: Deep Learning, Cloud Computing, Performance Benchmark, Virtualization, Container, Virtual Machine.

Abstract: The recent success of AI is contributed by the adaptation of using deep learning in decision making process. To harness the power of deep learning, developers must not only rely on a computing framework, but a cloud platform to ensure resource utilization and computing performance to ease the burden on users as well. Hence, ”*how cloud resources should be orchestrated for deep learning?*” becomes a fundamental question for cloud providers. In this work, we built an in-house OpenStack cloud platform to enable various resource orchestrations, including virtual machine, container and bare-metal. Then we systematically evaluate the performance of different orchestration choices using Tensorflow image classification benchmarks to quantify the performance impact and discuss the challenges of addressing these performance issues.

1 INTRODUCTION

Artificial Intelligent (AI) has been widely considered the next big thing for the future. According to the latest report (Gartner, 2017), AI is going to create $2.9 trillion of business value by 2021, furthermore, 41% of organizations in their research has already adopted AI solution. The recent success of AI is contributed by the adaptation of using deep learning in decision making process. Deep learning is a machine technique based on neural network (NN) model. By introducing multiple hidden layers in the NN model, and then learning the parameters from huge amount training datasets, it has been proven that the deep learning can achieve significant better prediction accuracy in various application domains, including image classification (Russakovsky et al., 2015), voice recognition (Noda et al., 2015), autonomous car (Ramos et al., 2017), etc. However, to harness the power of deep learning, application developers must rely on two things: a computing framework, and a resource pool.

A number of frameworks (Theano Development Team, 2016; Collobert et al., 2011; Chen et al., 2015; Abadi et al., 2015) have been developed for deep learning to ease the code development and execution management burden on users. Among them, Tensorflow (Abadi et al., 2015) is one of the most popular frameworks. At same time, deep learning consumes huge amount of computing capacity, because it often relies on deeper networks and larger training datasets to improve its model accuracy. Hence, some people starts to use the deep learning computing services offered by cloud providers to take advantage of the pay-as-you-used pricing model. Others decide to build their own private cloud infrastructure with more administration control and data privacy. In particularly, besides the traditional cloud infrastructure based on virtual machine, increasing attentions are drawn by the lightweight virtualization approach like container or even bare-metal. Moreover, according to (Bernstein, 2014), there are several other possible layering combinations for running containers and its runtime applications such as container-to-bare-metal, container-to-VM, and so on. Therefore, regardless which cloud model is used for accessing resource, or which framework is chosen for developing code, a fundamental problem is ”*how cloud resources should be orchestrated for deep learning?*”

To answer the aforementioned question, we built an in-house cloud platform using the open source cloud software, OpenStack (OpenStack, 2017). Necessary OpenStack service were installed to compare resource orchestration options: (1) bare-metal (BM); (2) virtual machine (VM); (3) container on BM; and (4) container on VM. We used a Tensorflow image

Lin, C., Pai, H. and Chou, J.
Comparison Between Bare-metal, Container and VM using Tensorflow Image Classification Benchmarks for Deep Learning Cloud Platform.
DOI: 10.5220/0006680603760383
In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, pages 376-383
ISBN: 978-989-758-295-0

classification benchmark suite to systematically evaluate the performance impacts and issues of using different orchestrations options under three training scenarios, including different input datasets (synthetic and real), different execution modes (single-host and distributed), and different resource environment (isolated and shared). Through a series of experimental studies using real application workloads, we quantified the performance overhead, and performance interference from resource orchestration, and identified the research challenges that deserved to receive more attentions in the future.

The rest of paper is structured as follows. Section 2 introduces the deep learning application and cloud service components used in our study. Section 3 describes our experimental methodology, and Section 4 summarizes the evaluation results. Related work is in Section 5. Section 6 concludes the paper.

2 BACKGROUND

2.1 Tensorflow

Tensorflow is one of the most popular deep learning frameworks. It provides a set of high-level API for users to build deep learning models more easily, and implements many features to optimize the deep learning computations on computing resources. For example, it supports parallel I/O to reduce the training data load time from disk and the data transfer time between CPU and GPU or between GPUs. Furthermore, distributed Tensorflow was introduced and released in 2016 to support deep learning computations across multiple nodes. In distributed Tensorflow, computations can be parallelized in several ways to reduce computation time. One of common approaches is to duplicate the model on each node and partition training dataset across nodes, so that each node can train the model with different data input in parallel. But since the model parameters must be synchronized among nodes, users can specify a set of parameter servers to store and update these shared model parameters throughout the training process with less communication traffic among nodes.

2.2 Compute Instances in OpenStack

In this work, we built a cloud platform using OpenStack, and installed the cloud service to orchestrate four types of resource orchestrations: (1) bare-metal (BM); (2) virtual machine (VM); (3) container on BM; and (4) container on VM. The architecture of our cloud platform is shown in Figure 1, and we

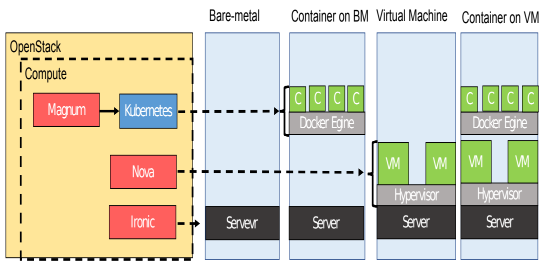

Figure 1: The OpenStack and its service components (i.e., Magnum, Nova and Ironic) that we installed to deploy the four types of compute instance based on virtual machine, container and bare-metal resource provisioning techniques.

briefly introduce the orchestration services used in the evaluation as follows.

Ironic (Bare-metal): A physical machine which is fully dedicated without any virtualization layer is expressed in terms of bare-metal. Ironic allocates the bare-metal as on-demand compute instance to users via PXE (Preboot eXecution Environment) and IPMI (Intelligent Platform Management Interface) mechanism. With bare-metal, applications in the cloud could be executed as in the local environment without losing performance due to the fully utilization of hardware. Meanwhile, the bare-metal still can be dynamically provisioned and allocated to users in a on-demand manner like other cloud resources. However, since a bare-metal is provisioned as a whole physical machine, the resource on it is dedicated to its users, and cannot be shared among other tenants in the cloud. Thus, bare-metal could result in lower resource utilization and compute instance deployment density in cloud.

Nova (Virtual Machine): Nova is the service in OpenStack to spawn and manage virtual machines through a VM hypervisor, such as QEMU. The virtualization of hardware resources allows a VM to be created with different resource configurations, and let multiple VMs to be run on one machine for sharing physical resources. Hence, comparing to bare-metal, virtual machine can increase the resource utilization, but introduce more performance overhead.

Magnum (Container): Container can be considered as lightweight virtualization. It isolates Linux processes into their own system environment, but shares the OS kernel with host machine. Hence, comparing to virtual machine, it has less performance overhead, higher deployment density, faster boot time. Thus, container has been widely used for software and service deployment. In OpenStack, containers are provisioned and managed through a container orchestration engine(COE). The service that interacts with COE is called Magnum, which provides an unified interface for users to control various COE implementations, including Kubernetes (Kubernetes, 2017), Swarm (Docker, 2017) and Mesos (Hindman et al., 2011). Magnum can launch COE on either virtual machines provisioned by Nova or bare-metal machines provisioned by Ironic. In this work, we choose Kubernetes as our COE.

3 METHODOLOGY

This section details our performance evaluation methodology. First, we give the hardware specification and software version in our experiments testbed. Then, we introduce the Tensorflow benchmark for generating the deep learning computation workload. Finally, we identify three commonly used execution scenarios of Tensorflow, and use them in our evaluation to reflect the actual application performance.

3.1 Testbed Environment

Our experiments were conducted on a in-house OpenStack cloud platform with 4 physical nodes connected by 1Gbps network. Each node equipped with four Intel(R) Core(TM) i5-6600 CPU, 62GB of RAM. Three of them owned two GeForce GTX 1080 GPUs, and they were used as the worker nodes for provisioning compute instance, including bare-metal, virtual machine and container. The Tensorflow jobs from our benchmark were also run on these worker nodes. The machine without GPU was used as the master node to deploy controller components, including the Kubernetes master.

How does each compute instance access GPU plays an important role in our works. For virtual machine, we use pci passthrough mechanism to enable VM access GPU without significantly losing performance. This is also how public cloud provider, like GCP and AWS, support GPU servers in their cloud platform. On the other hand, Kubernetes supports GPU through the kubelet deployed on every worker node. If a container requests for GPU resource, the kubelet initializes the container, and uses cgroup to map GPU devices onto the container.

Finally, the versions of software used in our deployment are summarized as follows. The OS version is CentOS 7.3.1611. The version of OpenStack is Mitaka. The version of Docker is v17.05.0-ce. The version of Kubernetes is v1.7.5. We use CUDA v8.0, cudnn v6.0 and Tensorflow-gpu v1.4 to execute all Tensorflow jobs in our experiments. As shown, all these software packages are released recently within a year or so.

Table 1: Configuration of models in our benchmark.

Options	InceptionV3	ResNet-50	AlexNet
Batch size/GPU	32	32	512
Data Format	NHWC		
Optimizer	sgd		
Variable update	Single node: parameter server Distributed: distributed replicated		
Local parameter device	Single node: CPU Distributed: GPU		

3.2 Image Classification Benchmarks

The benchmark applications used in our evaluation are three image classification CNN(Convolution Neural Network) models that won the ImageNet challenge in the past few years, including InceptionV3(Szegedy et al., 2015), ResNet-50(He et al., 2015) and AlexNet(Krizhevsky et al., 2012). These models can be either trained by a synthetic dataset or a real dataset (ImageNet dataset (Russakovsky et al., 2015)). Synthetic dataset is generated in memory and can avoid disk I/O overhead, while real dataset requires additional disk I/O operations to load data from file system. Noted, while real dataset is used, we replicate the dataset and place it on every node.

Each benchmark run is a model training job that does 10 warm-up steps followed by another 100 training steps. Our experiments focus on two key performance metrics: (1) *throughput* measured by the number of images processed per second during the training steps, and (2) *elapse time* measured by the total execution time from a training job is launched until it finished. In other words, the elapse time metric includes the performance impact of data loading and pre-processing while the throughput metric does not. All the configurations for running Tensorflow benchmark are summarized in Table 1.

3.3 Evaluation Scenarios

We design three evaluation scenarios in order to fully compare different compute environments and to capture behavioral performance of how the training jobs are commonly executed in a virtualized and shared cloud environment. The execution settings of these scenarios are described in below:

a) Single instance scenario: It represents the simplest setting for running a Tensorflow job. In this scenario, each job only runs on a single compute instance provisioned by the cloud resource orchestration manager. But in our experiments, each compute instance is attached with two GPU devices. Hence, we let the job to parallelize its computing workload across the two GPU devices, and launches a single parameter server process on CPU to coordinate the communication among them. This job execution scenario is also the most commonly seen approach by the Tensorflow users, because it can utilize all the GPU resource on a compute instance, and avoid communication overhead across network.

b) Distributed multi-instance scenario: It represents the setting when users need to perform large-scale model training. In this scenario, we use the distributed Tensorflow to launch a job running across dynamic provisioned compute instances. In the experiments, we uses three compute instances. One of them runs the Tensorflow parameter server for synchronization, and the other two runs the Tensorflow worker for training computations. The model parameters are distributed and replicated on GPU devices, and their values are updated after the parameter server collects the gradients from all workers. Each compute instance still has two GPU devices, so the computing workload is parallel distributed across both machine and device levels. This job execution scenario is often used to achieve shorter execution time or handle larger training model. However, due to the frequent message communications between workers and parameter servers from model training, significant network traffic could be incurred.

c) Shared resource scenario: To improve resource utilization and increase deployment density, it is common for cloud providers to consolidate multiple compute instances on the same physical machine. With the emergence of powerful GPU servers that can support up to 16 GPU cards, a single job does not often need all the resources. Therefore, our last scenario is to evaluate the impact of performance interference when multiple compute instances are created on the same physical machine for resource sharing. Our approach is to firstly run a job on a single compute instance to obtain its original performance measurements. Then we create two compute instances on the same physical machine, and run two jobs simultaneously on the instances (one job per compute instance) to observe the performance degradation and impact. Since each physical machine has only two GPU devices, each compute instance only uses one GPU device in this set of experiments. Noted for the case of bare-metal experiments, we simply run two jobs in the same OS environment.

4 EXPERIMENTAL RESULTS

This section summarizes the performance results of the three evaluation scenarios described in Section 3.3. The results in the following plots are normalized to bare-metal setting, and the actual performance measurement values are indicated above the bars.

4.1 Single Instance

This set of experiments evaluates the performance of running our benchmark on a single compute instance. First, we discuss the results from using synthetic data input. The results of training throughput, and the total job elapse time are shown in Figure 2, and Figure 3, respectively. As shown from the plots, bare-metal has the best performance, and container on virtual machine has the worst performance. The impact of virtual machine is larger than container as larger performance degradation is shown when virtual machine is used. The performance impact on throughput has the same trend as elapse time. But we found that the degradation is more apparent for elapse time. This is because elapse time includes the model copy time between CPU host and GPU device before and after training, and virtualization causes more stress on memory access than pure GPU or CPU computations. We also found that the impact on AlexNet is more than the other two models. It is likely because AlexNet has a larger training batch size which makes the memory copy performance matters more. But overall, the performance degradation in this set of experiments is limited within 15%, because both network and disk I/O operations were not involved when synthetic data input is used on a single compute instance.

The results of real data input are shown in Figure 4 and Figure 5. Comparing to the synthetic data input, real data input requires additional disk operations to load the training data from disk. Hence, the impact on disk I/O performance from virtualization is also included in this study. We found that container can still perform almost the same as bare-metal with real data input. However, the performance of virtual machine degrades much more significantly. For instance, the elapse time of container on VM becomes almost 30% longer than the bare-metal. In comparison, it is less than 20% for the synthetic data input. Overall, container performs similar to bare-metal in the single instance scenario with or without data input. But virtual machine may suffer more performance degradation when larger training datasets needed to be loaded from disks.

4.2 Distributed Multi-instance

This set of experiments evaluates the performance of running our benchmark across multiple compute instances which introduces additional network over-

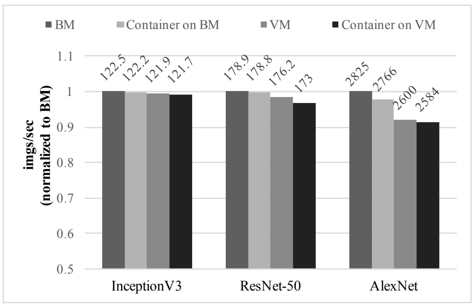

Figure 2: Images/sec throughput comparison using synthetic data on single instance. Lower values means higher performance degradation.

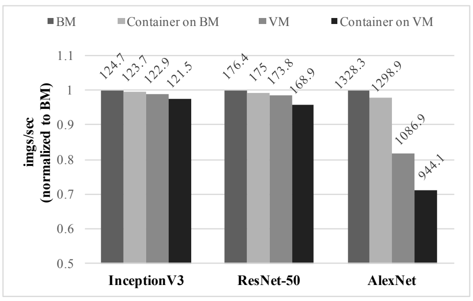

Figure 4: Images/sec throughput comparison using real data on single instance. Lower values means higher performance degradation.

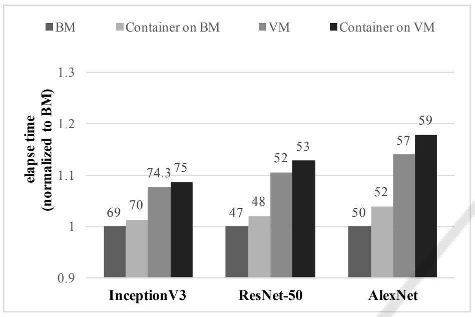

Figure 3: Job elapse time comparison using synthetic data on single instance. Higher values means higher performance degradation.

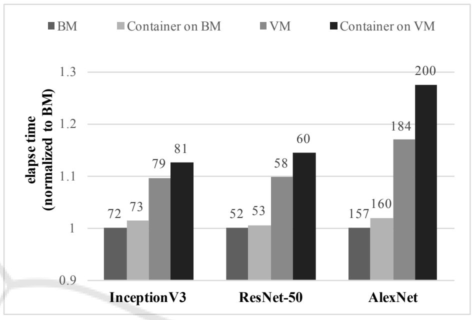

Figure 5: Job elapse time comparison using real data on single instance. Higher values means higher performance degradation.

head into our performance comparison. Again, we discuss the results of using synthetic data and real data, separately. The results of synthetic data are shown in Figure 6, and Figure 7. Comparing to the single instance scenario, more significant performance degradation is observed for the multi-instance scenario. As shown by the throughput comparisons of InceptionV3 and ResNet-50 models in Figure 6, the degradation of container on bare-metal already reaches about 20%. The degradation of virtual machine and container on virtual machines even reaches 25% and 35%, respectively. Therefore, container and virtual machine both cause significant network overhead. But interestingly, we found that network overhead has much less impact on AlexNet than the other two models. This is because AlexNet is much smaller model than the others, and thus it suffers less impact from the network performance degradation.

Figure 8 and Figure 9 are the results of using real data. Unlike single instance scenario, here we found similar results between the real data and synthetic data. This is because network performance dominates the overall performance for distributed Tensorflow, so the disk I/O impact was shadowed. Overall, we found that network performance is critical to distributed Tensorflow. Although container can deliver close to bare-metal computing performance, it also suffers severe network performance degradation like virtual machine. We believe it is caused by the additional network layer, flannel, added by Kubernetes. Finally, the elapse time can be prolonged by as much as 1.5 times longer when running on both container and virtual machine as shown in Figure 9. We expect the degradation to be even greater if the model is more complex or the train dataset is larger. Therefore, cloud providers must pay more attentions to network virtualization in the future.

4.3 Shared Resource Environment

Lastly, we evaluate the performance in a shared resource environment where multiple jobs running on the same physical machine simultaneously. We evaluate the performance degradation by comparing the job elapse time before and after the background workload is introduced. The background workload used in this set of experiments is a AlexNext training job with a batch size of 512. The results of using different compute instance settings are show in Figure 10. Noted, for the setting of bare-metal, we directly run two jobs in the same OS environment.

As expected, across all the test cases, bare mental has the highest performance impact in shared resource environment, while virtual machine has the

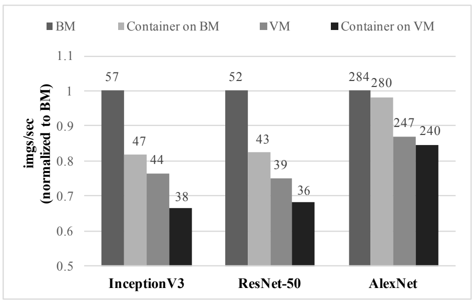

Figure 6: Images/sec throughput comparison of distributed Tensorflow using synthetic data on multiple instances. Lower values means higher performance degradation.

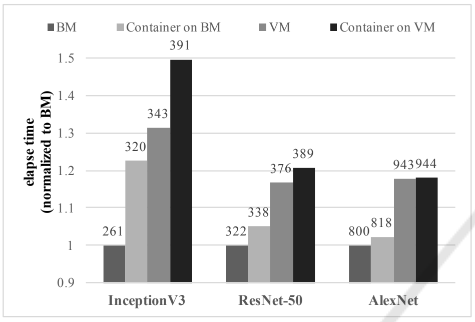

Figure 7: Job elapse time comparison of distributed Tensorflow using synthetic data on multiple instances. Higher values means higher performance degradation.

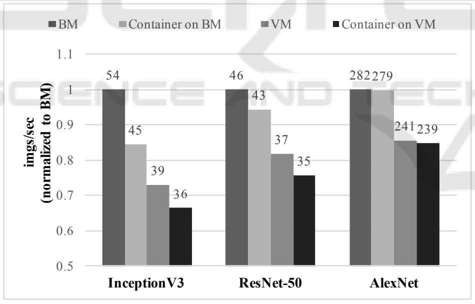

Figure 8: Images/sec throughput comparison of distributed Tensorflow using real data on multiple instances. Lower values means higher performance degradation.

least. Bare-metal has the highest performance impact because it does not limit resource usage between the processes from different jobs, and processes can be context switched among CPUs arbitrary. In comparison, containers are bound to user designated CPUs, and kubernetes also has QoS mechanism to control the resource usage of a container within a user specified range. Virtual machine offers the strongest resource isolation because jobs are managed by separate operating systems, and resources are reserved on host machines in advanced, so performance interference is minimized.

But interestingly, we observer that the performance impact is limited in most cases, except for AlexNext with 512 batch size. After further looking into the job profiling log, we found that the execution time delay was caused by the much longer memory copy duration between host and devices. We believe this is because larger batch size means more training data needs to be transferred to GPU in each training step. As a result, higher I/O bandwidth between GPU device and CPU host is required. Since the batch size of both InceptionV3 and ResNet-50 is only 32, there were still enough bandwidth to be shared between foreground and background jobs. But when we ran two AlexNext jobs with 512 batch size together, the required bandwidth has over the hardware I/O bus limit, and thus cause significant performance degradation. To verify our guess, we also evaluated the case of running AlexNet with 32 batch size setting. As shown by the rightmost bars in Figure 10, the performance is again under 10% for container and virtual machine.

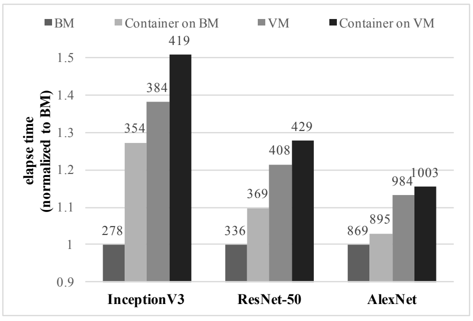

Figure 9: Job elapse time comparison of distributed Tensorflow using real data on multiple instances. Higher values means higher performance degradation.

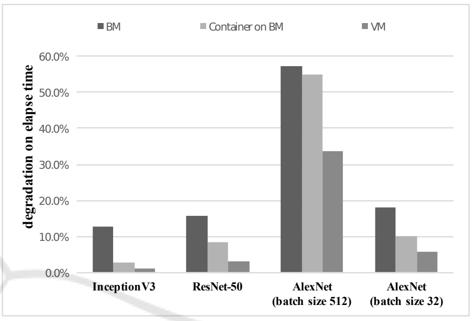

Figure 10: Performance degradation of job elapse time in shared resource environment. The baseline performance is measured when background workload is not introduced.

To sum up, with smaller batch size, we found that the performance degradation is within 15%∼20% regardless what type of compute instances is used because most computation workload is located on GPU devices not CPU devices. However, when a larger batch size is used, it can cause resource contention on the I/O bus between host and devices, and this problem has not been addressed by the existing virtualization technologies. Therefore, under resource sharing environment, we only not need to provision resource usage on GPU and CPU, but also the I/O bandwidth between GPU and CPU.

5 RELATED WORKS

Many studies have conducted experiments to compare the three types of virtualized resource instances (bare-metal, virtual machine and container) in different computing environment and using various benchmarks. We summarize some of them in below.

Firstly, there are plenty performance comparison studies between container and VM. (Salah et al., 2017) compares the two from the services and microservice architecture perspective by evaluating the service deployment time of using container service (ECS) and virtual machine service (EC2) in AWS cloud platform. Although the ECS containers are actually running on EC2 virtual machines, its deployment time still much shorter than EC2 because AWS can re-use active virtual machine for container deployment. (Li et al., 2017) compares the computing performance between VM and container by various different types of applications. Their results show that although the container-based solution is undoubtedly lightweight, the VM does not always have worse performance. For instance, they did observe that container has slower transaction speed than VM for bytes level data read/write operations. (Jlassi and Martineau, 2016) benchmarks Hadoop on Docker containers, VMware and heterogeneous cluster consisting of both. It shows that container has better performance in most experiments, but the energy consumption could be varied according to the executed workload.

Then there are studies considering container as a deployment service layer on top of resource virtualization layer. Hence, (Ruan et al., 2016) conducted a series of experiments to measure performance differences between application containers (e.g., Docker) and system containers (e.g., LXC), then evaluate the overhead of extra virtual machine layer, and finally inspect the service quality of container service in AWS and Google Cloud Platform.

Finally, both (Kominos et al., 2017) and (Mazaheri et al., 2016) directly compare the performance between bare-metal, virtual machine and container. Their goal is to quantify the overhead of CPU, networking, disk I/O, RAM and boot-up time using various different high-performance computing benchmarks, such as HPCC (High Performance Computing Challenge) and IOR (Interleave Or Random). They both conclude that Docker container surpasses virtual machine in most benchmarks, and has merely the same performance of bare-metal.

All aforementioned studies aim to use singe resource-bound benchmark applications to compare the performance of using a specific type of resources, such as I/O, network, CPU. But deep learning is a rather complex application with mixed types of resource usage. Without directly using the real deep learning application workloads, it is difficult to characterize the performance impact from various training job settings, such as different batch size and network models, etc. Besides, in this paper, we compare four different orchestration options by using the combination of different virtualization techniques, and focus on the performance overhead and performance interference issues in a shared cloud environment.

6 CONCLUSION

Building cost effective and performance efficient cloud service for deep learning is an urgent matter. In this work, we aim to discuss the resource orchestration choices between bare-metal, container and virtual machine. We use OpenStack to build a cloud platform to evaluate the performance of these orchestration approaches by training a set image classification CNN models. We conclude the key findings from our experimental study as follows.

(1) On a single compute instance when network and disk I/O doesn't involve, virtualization layer has little impact to performance regardless which technique is used. Even when both container and virtual machine are used, the degradation we observed was less than 10%. Hence, virtualization overhead is not a critical concern in this use scenario.

(2) Distributed Tensorflow is a network bound application. Unfortunately, virtualization layer does cause significant degradation to network performance, especially for virtual machine. The network overhead of container is caused by the additional network overlay, flannel, in kubernetes. Hence, running container on virtual can exacerbate the performance overhead, and reach more than 35% throughput degradation, and 50% elapse time increment. Hence, lightweight virtualization technique like container or even bare-metal should be considered.

(3) In a shared resource environment, we found that there is limited resource contention problem on the host machines because Tensorflow offloads most of its computation workload to GPUs not CPUs. However, the resource contention on the I/O band-

width between host and device is a major concern, especially when larger batch size is used to transfer more data simultaneously. Our experiments show the degradation can lead to more than 30% elapse time increment, and this problem is neither addressed by container or virtual machine. Hence, cloud manager must take this resource usage requirements into account when running multiple jobs on the same machine.

(4) Finally, the actual severity of performance impact can be varied according to the train model characteristics. If the model is more complex, the network overhead becomes more important. If the training dataset becomes larger, the I/O or memory access overhead becomes more critical. But overall, it is better to use more lightweight virtualization or resource orchestration approach, and prevent additional virtualization layers.

Besides offering researchers more understanding of the resource orchestration impact on deep learning applications, we identify the following research challenges that deserved to receive more attentions in the future: (1) improve network virtualization performance and reduce overlay network layers in resource orchestration; (2) provide resource sharing and controlling mechanism on a single GPU device as well as the I/O bandwidth resource between devices and hosts; (3) develop more accurate resource usage estimation and performance prediction mechanism for deep learning job to help cloud providers optimize their job scheduling and placement decision.

REFERENCES

Abadi, M., Agarwal, A., and et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bernstein, D. (2014). Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Chen, T., Li, M., and et al. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

Docker (2017). Docker swarm. https://docs.docker.com/engine/swarm/.

Gartner (2017). Gartner. http://www.gartner.com/.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Hindman, B., Konwinski, A., and et al. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308.

Jlassi, A. and Martineau, P. (2016). Benchmarking hadoop performance in the cloud - an in depth study of resource management and energy consumption. In *International Conference on Cloud Computing and Services Science*, pages 192–201.

Kominos, C. G., Seyvet, N., and Vandikas, K. (2017). Bare-metal, virtual machines and containers in openstack. In *20th ICIN*, pages 36–43.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *25th NIPS*, volume 1, pages 1097–1105.

Kubernetes (2017). Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. https://kubernetes.io/.

Li, Z., Kihl, M., Lu, Q., and Andersson, J. A. (2017). Performance overhead comparison between hypervisor and container based virtualization. In *IEEE AINA*, pages 955–962.

Mazaheri, S., Chen, Y., Hojati, E., and Sill, A. (2016). Cloud benchmarking in bare-metal, virtualized, and containerized execution environments. In *IEEE CCIS*, pages 371–376.

Noda, K., Yamaguchi, Y., and et al. (2015). Audio-visual speech recognition using deep learning. *Appl. Intell.*, 42(4):722–737.

OpenStack (2017). Open source software for creating private and public clouds. https://www.openstack.org/.

Ramos, S., Gehrig, S. K., Pinggera, P., Franke, U., and Rother, C. (2017). Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling. In *IEEE Intelligent Vehicles Symposium*, pages 1025–1032.

Ruan, B., Huang, H., Wu, S., and Jin, H. (2016). A performance study of containers in cloud environment. In *Asia-Pacific Services Computing Conference*, volume 10065 of *Lecture Notes in Computer Science*, pages 343–356.

Russakovsky, O., Deng, J., and et al. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., and Al-Hammadi, Y. (2017). Performance comparison between container-based and vm-based services. In *19th ICIN*, pages 185–190.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.