Dell EMC Ready Solutions for AI – Deep Learning with NVIDIA

Deep Learning with NVIDIA Architecture Guide

Authors: Rengan Xu, Frank Han, Nishanth Dandapanthula

Abstract

There has been an explosion of interest in Deep Learning and the plethora of choices makes designing a solution complex and time consuming. Dell EMC's Ready Solutions for AI – Deep Learning with NVIDIA is a complete solution, designed to support all phases of Deep Learning, incorporates the latest CPU, GPU, memory, network, storage, and software technologies with impressive performance for both training and inference phases. The architecture of this Deep Learning solution is presented in this document.

August 2018

Revisions

Date	Description
August 2018	Initial release

Table of Contents

Executive summary

Deep Learning techniques has enabled great success in many fields such as computer vision, natural language processing (NLP), gaming and autonomous driving by enabling a model to learn from existing data and then to make corresponding predictions. The success is due to a combination of improved algorithms, access to large datasets and increased computational power. To be effective at enterprise scale, the computational intensity of Deep Learning neural network training requires highly powerful and efficient parallel architectures. The choice and design of the system components, carefully selected and tuned for Deep Learning use-cases, can make the difference in the business outcomes of applying Deep Learning techniques. In addition to several options for processors, accelerators and storage technologies, there are multiple Deep Learning software frameworks and libraries that must be considered. These software components are under active development, updated frequently and cumbersome to manage. It is complicated to simply build and run Deep Learning applications successfully, leaving little time for focus on the actual business problem.

To resolve this complexity challenge, Dell EMC has developed an architecture for Deep Learning that provides a complete, supported solution. This solution includes carefully selected technologies across all aspects of Deep Learning, processing capabilities, memory, storage and network technologies as well as the software ecosystem. This document presents the architecture of this Deep Learning solution including details on the design choice for each component. The performance aspects of this complete solution have also been characterized and are also described here.

AUDIENCE

This document is intended for organizations interested in accelerating Deep Learning with advanced computing and data management solutions. Solution architects, system administrators and others interested readers within those organizations constitute the target audience.

1 Solution Overview

Dell EMC has developed an architecture for Deep Learning that provides a complete, supported solution. This solution includes carefully selected technologies across all aspects of Deep Learning, processing capabilities, memory, storage and network technologies as well as the software ecosystem. This complete solution is provided as Dell EMC's Ready Solutions for AI – Deep Learning with NVIDIA. The solution includes fully integrated and optimized hardware, software, and services including deployment, integration and support making it easier for organizations to start and grow their Deep Learning practice.

The high level overview of Dell EMC Ready Solutions for AI - Deep Learning is shown in Figure 1.

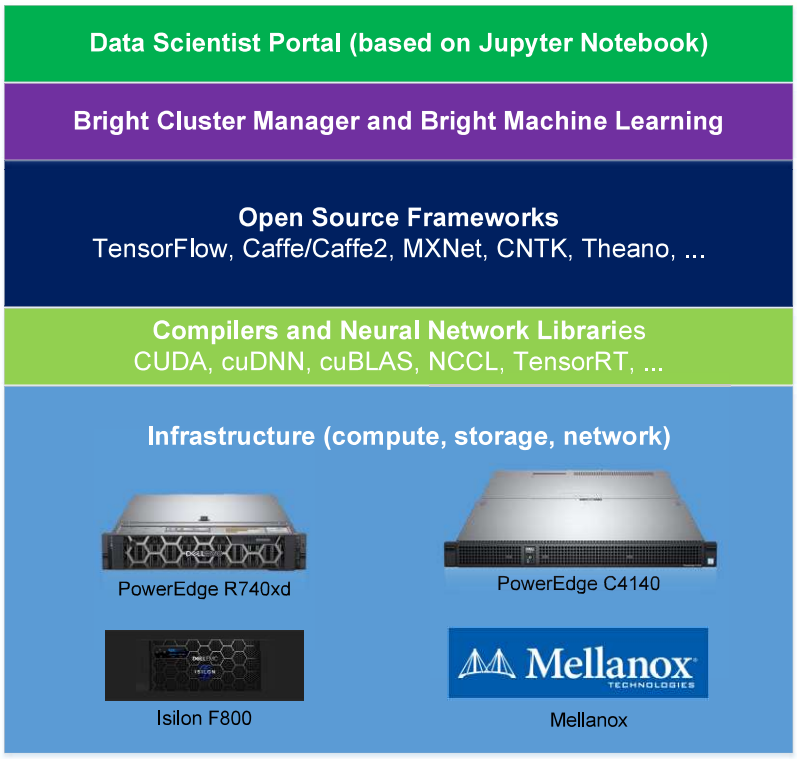


Figure 1: Overview of Dell EMC Ready Solutions for AI - Deep Learning

Data Scientist Portal: This is a new portal for data scientists created for this solution. It enables data scientists, who should not need to be experts in cluster technologies, to use a simple web portal to take advantage of the underlying technology. The scientists can write, train and do inference for different Deep Learning models within Jupyter Notebook which includes Python 2, Python 3, R and other kernels.

Bright Cluster Manager and Bright Machine Learning: Bright Cluster Manager is used for the monitoring, deployment, management, and maintenance of the cluster. The Bright Machine Learning (ML) includes the deep learning frameworks, libraries, and compilers and so on.

Deep Learning Frameworks and Libraries: This category includes TensorFlow, MXNet, Caffe2, CUDA, cuDNN, and NCCL. The latest version of these frameworks and libraries are integrated into the solution.

Infrastructure: The infrastructure comprises of a cluster with a master node, compute nodes, shared storage and networks. In this instance of the solution, the master node is a Dell EMC PowerEdge R740xd, each compute node is PowerEdge C4140 with NVIDIA Tesla GPUs, the storage includes Network File System (NFS) and Isilon, and the networks include Ethernet and Mellanox InfiniBand.

Section 2 describes each of these solution components in more detail, covering the compute, network, storage and software configurations. Extensive performance analysis on this solution was conducted in the HPC and AI Innovation Lab and those results are presented in Section 3. These includes tests with training and inference workloads, conducted on different types of GPUs, using different floating point and integer precision arithmetic, and with different storage sub-systems for Deep Learning workloads. That is followed by Section 4 that describes containerization techniques for Deep Learning. Section 5 has details on the Data Scientist Portal developed by Dell EMC. Conclusion and future direction completes the document in Section 6.

2 Solution Architecture

The hardware comprises of a cluster with a master node, compute nodes, shared storage and networks. The master node or head node roles can include deploying the cluster of compute nodes, managing the compute nodes, user logins and access, providing a compilation environment, and job submissions to compute nodes. The compute nodes are the work horse and execute the submitted jobs. Software from Bright Computing called Bright Cluster Manager is used to deploy and manage the whole cluster.

Figure 2 shows the high-level overview of the cluster which includes one head node, *n* compute nodes, the local disks on the cluster head node exported over NFS, Isilon storage, and two networks. All compute nodes are interconnected through an InfiniBand switch. The head node is also connected to the InfiniBand switch as it uses IPoIB to export the NFS share to the compute nodes. All compute nodes and the head node are also connected to a 1 Gigabit Ethernet management switch which is used by Bright Cluster Manager to administer the cluster. An Isilon storage solution is connected to the FDR-40GigE Gateway switch so that it can be accessed by the head node and all compute nodes.

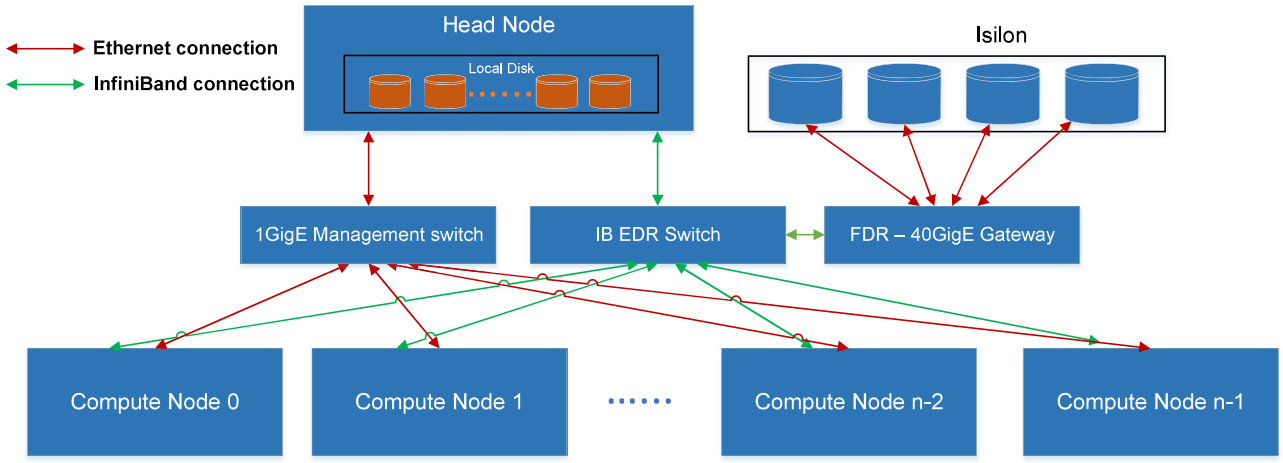


Figure 2: The overview of the cluster

2.1 Head Node Configuration

The Dell EMC PowerEdge R740xd is recommended for the role of the head node. This is Dell EMC's latest two socket, 2U rack server that can support the memory capacities, I/O needs and network options required of the head node. The head node will perform the cluster administration, cluster management, NFS server, user login node and compilation node roles.

The suggested configuration of the PowerEdge R740xd is listed in Table 1. It includes 12 x 12TB NL SAS local disks that are formatted as an XFS file system and exported via NFS to the compute nodes over IPoIB. RAID 50 is used instead of RAID6/RAID60 to take into consideration faster rebuild time and capacity advantages provided by the former. Details of each configuration choice are described in the following sections. For more information on this server model please refer to PowerEdge R740/740xd Technical Guide.

Table 1: PowerEdge R740xd configurations

Component	Details
Server Model	PowerEdge R740xd
Processor	2 x Intel Xeon Gold 6148 CPU @ 2.40GHz
Memory	24 x 16GB DDR4 2666MT/s DIMMs - 384GB
Disks	12 x 12TB NL SAS RAID 50 (Recommended 10+ drives)
I/O & Ports	Network daughter card with 2 x 10GE + 2 x 1GE
Network Adapter	1x InfiniBand EDR adapter
Out of Band Management	iDRAC9 Enterprise with Lifecycle Controller
Power Supplies	Titanium 1100W, Platinum
Storage Controllers	PowerEdge RAID Controller (PERC) H730p

2.1.1 Shared Storage via NFS over InfiniBand

The default shared storage system for the cluster is provided over NFS. It is built using 12x 12 TB NL SAS disks that are local to the head node configured in RAID 50 with two parity check disks. This provides usable capacity of 120TB (109TiB). RAID 50 was chosen because it has balanced performance and shorter rebuild time compared to RAID 6 or RAID 60 (since RAID 50 has fewer parity disks than RAID 6 or RAID 60). This 120TB volume is formatted as an XFS file system and exported to the compute nodes via NFS over IPoIB.

In the default configuration, both home directories and shared application and library install locations are hosted on this NFS share. In addition to this, for solutions which require a larger capacity shared storage solution, the Isilon F800 is as an alternative option and is described in Section 2.5. A comparison between various storage subsystems is provided in Section 3.1.5, including this NL SAS NFS, the Isilon, and smaller test configurations using SSDs and NVMe devices.

2.2 Compute Node Configuration

Deep Learning methods would not have gained success without the computational power to drive the iterative training process. Therefore, a key component of Deep Learning solutions is highly capable nodes that can support compute intensive workloads. The state-of-art neural network models in Deep Learning have more than 100 layers which require the computation to be able to scale across many compute nodes in order for any timely results. The Dell EMC PowerEdge C4140, an accelerator-optimized, high density 1U rack server, is used as the compute node unit in this solution. The PowerEdge C4140 can support four NVIDIA Volta SMX2 GPUs, both the V100-SXM2 as well as the V100-PCIe models. Figure 3 shows the CPU-GPU and GPU-GPU connection topology of a compute node.

The detailed configuration of each PowerEdge C4140 compute node is listed in Table 2.

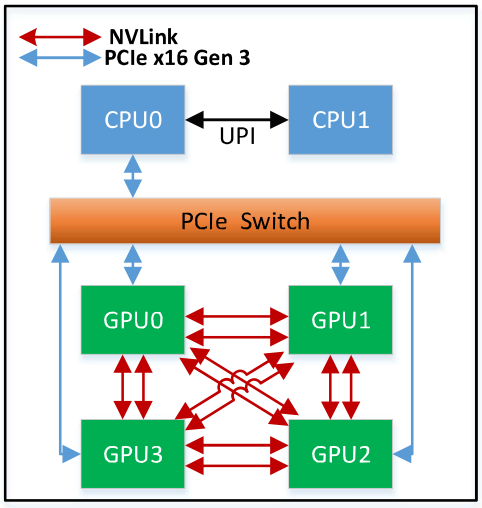


Figure 3: The topology of a compute node

Table 2: PowerEdge C4140 Configurations

Component	Details
Server Model	PowerEdge C4140
Processor	2 x Intel Xeon Gold 6148 CPU @ 2.40GHz
Memory	24 x 16GB DDR4 2666MT/s DIMMs - 384GB
Local Disks	120GB SSD, 1.6TB NVMe
I/O & Ports	Network daughter card with 2 x 10GE + 2 x 1GE
Network Adapter	1 x InfiniBand EDR adapter
GPU	4 x V100-SXM2 16GB
Out of Band Management	iDRAC9 Enterprise with Lifecycle Controller
Power Supplies	2000W hot-plug Redundant Power Supply Unit (PSU)

2.2.1 GPU

The NVIDIA Tesla V100 is the latest data center GPU available to accelerate Deep Learning. Powered by NVIDIA Volta™, the latest GPU architecture, Tesla V100 GPUs enable data scientists, researchers, and engineers to tackle challenges that were once difficult. With 640 Tensor Cores, Tesla V100 is the first GPU to break the 100 teraflops (TFLOPS) barrier of Deep Learning performance.

Table 3: V100-SXM2 vs V100-PCIe

Description	V100-PCIe	V100-SXM2
CUDA Cores	5120	5120
GPU Max Clock Rate (MHz)	1380	1530

Tensor Cores	640	640
Memory Bandwidth (GB/s)	900	900
NVLink Bandwidth (GB/s) (uni-direction)	N/A	300
Deep Learning (Tensor OPS)	112	120
TDP (Watts)	250	300

Tesla V100 product line includes two variations, V100-PCIe and V100-SXM2. The comparison of two variants is shown in Table 3. In the V100-PCIe, all GPUs communicate with each other over PCIe buses. With the V100-SXM2 model, all GPUs are connected by NVIDIA NVLink. In use-cases where multiple GPUs are required, the V100-SXM2 models provide the advantage of faster GPU-to-GPU communication over the NVLINK interconnect when compared to PCIe. V100-SXM2 GPUs provide six NVLinks per GPU for bi-directional communication. The bandwidth of each NVLink is 25GB/s in uni-direction and all four GPUs within a node can communicate at the same time, therefore the theoretical peak bandwidth is 6*25*4=600GB/s in bi-direction. However, the theoretical peak bandwidth using PCIe is only 16*2=32GB/s as the GPUs can only communicate in order, which means the communication cannot be done in parallel. So in theory the data communication with NVLink could be up to 600/32=18x faster than PCIe. The evaluation of this performance advantage in real models will be discussed in Section 3.1.3. Because of this advantage, the PowerEdge C4140 compute node in the Deep Learning solution uses V100-SXM2 instead of V100-PCIe GPUs.

2.3 Processor recommendation for Head Node and Compute Nodes

The processor chosen for the head node and compute nodes is Intel® Xeon® Gold 6148 CPU. This is the latest Intel® Xeon® Scalable processor with 20 physical cores which support 40 threads. Previous studies, as described in Section 3.1, have concluded that 16 threads are sufficient to feed the I/O pipeline for the state-of-the-art convolutional neural network, so the Gold 6148 CPU is a reasonable choice. Additionally this CPU model is recommended for the compute nodes as well, making this a consistent choice across the cluster.

2.4 Memory recommendation for Head Node and Compute Nodes

The recommended memory for the head node is 24x 16GB 2666MT/s DIMMs. Therefore the total size of memory is 384GB. This is chosen based on the following facts:

- Capacity: An ideal configuration must support system memory capacity that is larger than the total size of GPU memory. Each compute node has 4 GPUs and each GPU has 16GB memory, so the system memory must be at least 16GBx4=64GB. The head node memory also affects I/O performance. For NFS service, larger memory will reduce disk read operations since NFS service needs to send out data from memory. 16GB DIMMs demonstrate the best performance/dollar value.
- DIMM configuration: Choices like 24x 16GB or 12x 32GB will provide the same capacity of 384GB system memory, but according to our studies as shown in Figure 4, the combination of 24x 16GB DIMMs provides 11% better performance than using 12x 32GB. The results shown here was on the Intel Xeon Platinum 8180 processor, but the same trends will apply across other models in the Intel Scalable Processor Family including the Gold 6148, although the actual percentage differences across configurations may vary. More details can be found in our Skylake memory study.
- Serviceability: The head node and compute nodes memory configurations are designed to be similar to reduce parts complexity while satisfying performance and capacity needs. Fewer parts need to be stocked for replacement, and in urgent cases if a memory module in the head node needs to be

replaced immediately, a DIMM module from a compute node can be temporarily considered to restore the head node until replacement modules arrive.

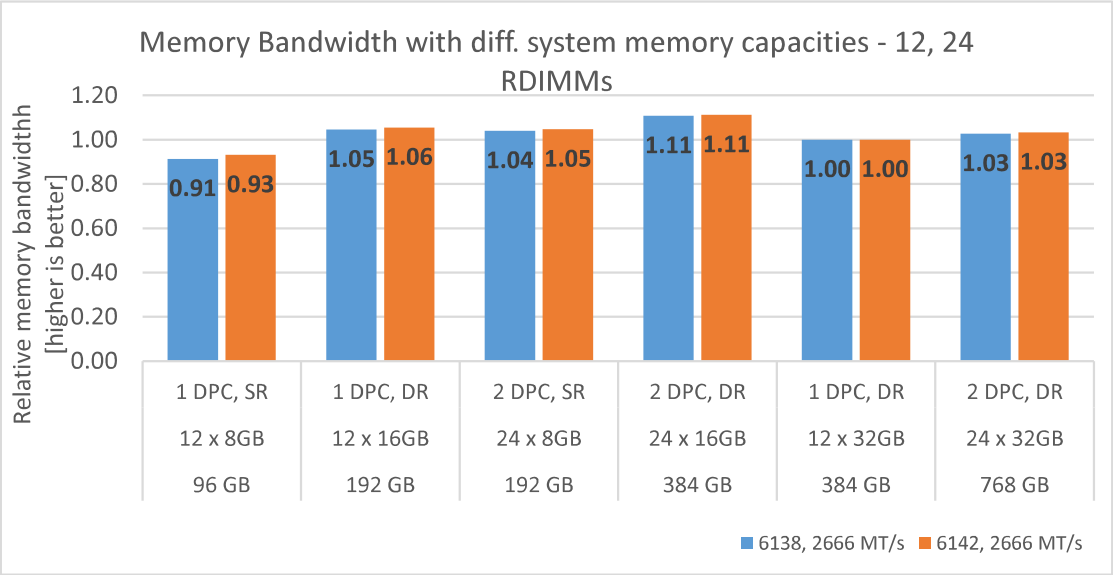


Figure 4: Relative memory bandwidth for different system capacities

2.5 Isilon Storage

Dell EMC Isilon is a proven scale-out network attached storage (NAS) solution that can handle the unstructured data prevalent in many different workflows. The Isilon storage architecture automatically aligns application needs with performance, capacity, and economics. As performance and capacity demands increase, both can be scaled simply and non-disruptively, allowing applications and users to continue working.

Dell EMC Isilon OneFS operating system powers all Dell EMC Isilon scale-out NAS storage solutions and has the following features.

- A high degree of scalability, with grow-as-you-go flexibility
- High efficiency to reduce costs
- Multi-protocol support such as SMB, NFS, HTTP and HDFS to maximize operational flexibility
- Enterprise data protection and resiliency
- Robust security options

The recommended Isilon storage is Isilon F800 all-flash scale-out NAS storage. Dell EMC Isilon F800 all-flash Scale-out NAS storage is uniquely suited for modern Deep Learning applications delivering the flexibility to deal with any data type, scalability for data sets ranging in the PBs, and concurrency to support the massive concurrent I/O request from the GPUs. Isilon's scale-out architecture eliminates the I/O bottleneck between storage and compute, allowing you to start with 10's of TB's of data with up to 15 GB/s bandwidth and then you can scale-out up to 68 PB with up to 540 GB/s of performance in a single cluster. This allows Isilon to accelerate AI innovation with faster model training, provide more accurate insights with deeper data sets, and deliver a higher ROI by fully saturating the data requests of up to 1000's of GPU's per cluster.

The Isilon storage can be used if the local NFS storage capacity is insufficient for the environment. If the Isilon is used in conjunction with the local NFS storage, user home directories and project results can be stored on

the Isilon with applications installed on the local NFS. The performance comparison between Isilon and other storage solutions are shown in Section 3.1.6. The specifications of the Isilon F800 are listed in Table 4.

Table 4: Specification of Isilon F800

Storage			
External storage	Isilon F800 All-flash Scale-out NAS		
Bandwidth	15 GB/s per chassis, Scales to 540 GB/s per cluster		
IOPS	250,000 IOPS per chassis, Scales to 9 Million IOPS per cluster		
Chassis Capacity (4 RU)	1.6 TB SSD x 60	3.2 TB SSD x 60	15.4 TB SSD x 60
	96 TB	192 TB	924 TB
Cluster Capacity	All-Flash: 96TB up to 33 PB; Hybrid: 96TB up to 68 PB		
Network	8x 40GbE (QSFP+) per chassis, (2x per node)		

Before doing deep learning model training, if a user wants to move very large data outside the cluster described in Section 2 to Isilon, the user can connect the server which stores the data to the FDR-40GigE gateway in Figure 2, so that the data can be moved onto Isilon without having to route it through the head node.

To monitor and analyze the performance and file system of Isilon storage, the tool InsightIQ can be used. InsightIQ allows a user to monitor and analyze Isilon storage cluster activity using standard reports in the InsightIQ web-based application. The user can customize these reports to provide information about storage cluster hardware, software, and protocol operations. InsightIQ transforms data into visual information that highlights performance outliers, and helps users diagnose bottlenecks and optimize workflows. In Section 3.1.5, InsightIQ was used to collect the average disk operation size, disk read IOPS, and file system throughput when running deep learning models. For more details about InsightIQ, refer to Isilon InsightIQ User Guide.

2.6 Network

The solution comprises of three network fabrics. The head node and all compute nodes are connected with a 1 Gigabit Ethernet fabric. The Ethernet switch recommended for this is the Dell Networking S3048-ON which has 48 ports. This connection is primarily used by Bright Cluster Manager for deployment, maintenance and monitoring the solution.

The second fabric connects the head node and all compute nodes are through 100 Gb/s EDR InfiniBand. The EDR InfiniBand switch is Mellanox SB7800 which has 36 ports. This fabric is used for IPC by the applications as well as to serve NFS from the head node (IPoIB) and Isilon. GPU-to-GPU communication across servers can use a technique called GPUDirect Remote Direct Memory Access (RDMA) which is enabled by InfiniBand. This enables GPUs to communicate directly without the involvement of CPUs. Without GPUDirect, when GPUs across servers need to communicate, the GPU in one node has to copy data from its GPU memory to system memory, then that data is sent to the system memory of another node over the network, and finally the data is copied from the system memory of the second node to the receiving GPU's memory. With GPUDirect however, the GPU on one node can send the data directly from its GPU memory to the GPU memory in another node, without going through the system memory in both nodes. Therefore GPUDirect decreases the GPU-GPU communication latency significantly.

The third switch in the solution is called a gateway switch in Figure 2 and connects the Isilon F800 to the head node and compute nodes. Isilon's external interfaces are 40 Gigabit Ethernet. Hence, a switch which can serve as the gateway between the 40GbE Ethernet and InfiniBand networks is needed for connectivity to the head and compute nodes. The Mellanox SX6036 is used for this purpose. The gateway is connected to the InfiniBand EDR switch and the Isilon as shown in Figure 2.

2.7 Software

The software portion of the solution is provided by Dell EMC and Bright Computing. The software includes several pieces.

The first piece is Bright Cluster Manager which is used to easily deploy and manage the clustered infrastructure and provides all cluster software including the operating system, GPU drivers and libraries, InfiniBand drivers and libraries, MPI middleware, the Slurm schedule, etc.

The second piece is the Bright machine learning (ML) which includes any deep learning library dependencies to the base operating system, deep learning frameworks including Caffe/Caffe2, Pytorch, Torch7, Theano, Tensorflow, Horovod, Keras, DIGITS, CNTK and MXNet, and deep learning libraries including cuDNN, NCCL, and the CUDA toolkit.

The third piece is the Data Scientist Portal which was developed by Dell EMC. The portal was created to abstract the complexity of the deep learning ecosystems by providing a single pane of glass which provides users with an interface to get started with their models. The portal includes spawner for Jupyterhub and integrates with

- Resource managers and schedulers (Slurm)
- LDAP for user management
- Deep Learning framework environments (Tensor Flow, Keras, MXNet, Pytorch etc.) from Bright's module environment, Python2, Python3 and R kernel support
- Tensorboard
- Terminal CLI environments.

It also provides templates to get started with for various DL environments and adds support for singularity containers. For more details about how to use the Data Scientist Portal, refer to Section 5.

3 Deep Learning Training and Inference Performance and Analysis

In this section, the performance of Deep Learning training as well as inference is measured using three open source Deep Learning frameworks: TensorFlow , MXNet and Caffe2 . The experiments were conducted on an instance of the solution architecture described in Section 2. The experiment test cluster used a PowerEdge R740xd head node, and PowerEdge C4140 compute nodes, different storage sub-systems including Isilon and InfiniBand EDR network. A detailed test bed description is provided in the following section.

3.1 Deep Learning Training

The well-known ILSVRC2012 dataset was used for benchmarking performance. This dataset contains 1,281,167 training images and 50,000 validation images in 140GB. All images are grouped into 1000 categories or classes. The overall size of ILSVRC 2012 leads to non-trivial training times and thus makes it more interesting for analysis. Additionally this dataset is commonly used by Deep Learning researchers for benchmarking and comparison studies. Resnet50 is a computationally intensive network and was selected to stress the solution to its maximum capability. For the batch size parameter in Deep Learning, the maximum batch size that does not cause memory errors was selected; this translated to a batch size of 64 per GPU for MXNet and Caffe2, and 128 per GPU for TensorFlow. Horovod, a distributed TensorFlow framework, was used to scale the training across multiple compute nodes. Throughput this document, performance was measured using a metric of images/sec which is a measure of throughput of how fast the system can complete training the dataset.

The images/sec result was averaged across all iterations to take into account the deviations. The total number of iterations is equal to num_epochs*num_images/(batch_size*num_gpus), where num_epochs means the number of passes to all images of a dataset, num_images means the total number of images in the dataset, batch_size means the number of images that are processed in parallel by one GPU, and num_gpus means the total number of GPUs involved in the training.

Before running any benchmark, the cache on the head node and compute node(s) were cleared with the command "`sync; echo 3 > /proc/sys/vm/drop_caches`". The training tests were run for a single epoch, or one pass through the entire dataset, since the throughput is consistent through epochs for MXNet and TensorFlow tests. Consistent throughput means that the performance variation was not significant across iterations, the tests measured less than 2% variation in performance.

However, two epochs were used for Caffe2 as it needs two epochs to stabilize the performance. This is because Caffe2's implementation is different than other two frameworks. Its training performance (throughput or images/sec) is not stable (the performance variation between iterations is large) when the dataset is not fully loaded in memory.

For MXNet framework, 16 CPU threads were used for dataset decoding and the reason was explained in the blog "Deep Learning on V100". Caffe2 does not provide a parameter for users to set the number of CPU threads. For TensorFlow, the number of CPU threads used for dataset decoding is calculated by subtracting four threads per GPU from the total physical core count of the system. The four threads per GPU are used for GPU compute, memory copies, event monitoring, and sending/receiving tensors. The processor used in these tests has 20 cores, with 40 cores per server. Hence 24 (40 threads – 4 threads/GPU * 4 GPUs) threads were used for data decoding per compute node for the TensorFlow tests. Table 5 lists the hardware and software details of the testbed used for the results presented in the following sections.

Table 5: The hardware and software in the testbed

Hardware – Head node	
Cluster head node	PowerEdge R740xd
CPU	2 x Intel Xeon 6148 @ 2.4GHz
Memory	384GB DDR4 @ 2667MT/s
Disks on head node	12 x12 TB Near-line SAS drives in a RAID 50 volume. 120TB volume formatted as XFS, exported via NFS
Hardware – Compute node	
Cluster compute node	PowerEdge C4140
Number of compute nodes	8 nodes with V100-PCIe and 2 nodes with V100-SXM2
CPU	2 x Intel Xeon 6148 @ 2.4GHz
Memory	384GB DDR4 @ 2667MT/s
Disks	2x M.2, 240GB in Raid1
GPU	V100-SXM2, V100-PCIe
Software and Firmware	
Operating System	Red Hat Enterprise Linux 7.4
Linux Kernel	3.10.0-693.el7.x86_64
BIOS	1.1.6
CUDA compiler and GPU driver	CUDA 9.1.85 (390.46)
Python	2.7.5
Deep Learning Datasets	
Dataset for training	ILSVRC2012 training dataset, 1,281,167 images
Dataset for inference	ILSVRC2012 validation dataset, 50,000 images
Deep Learning Libraries and Frameworks	
CUDNN	7.0
NCCL	2.1.15
Horovod	0.12.1
TensorFlow	1.8
MXNet	0.11.1
Caffe2	0.8.1+
TensorRT	4.0.0.3

3.1.1 FP16 vs FP32

The V100 GPUs contains a new type of processing core called Tensor Cores which support mixed precision training. Although many High Performance Computing (HPC) applications require high precision computation with FP32 (32-bit floating point) or FP64 (64-bit floating point), Deep Learning researchers have found they are able to achieve the same inference accuracy with FP16 (16-bit floating point) as can be had with FP32. In this

document, mixed precision training which includes FP16 and FP32 representations is denoted as “FP16” training. This section compares the performance of using FP16 for training versus FP32.

In experiments where training tests were executed using FP16 precision, the batch size was doubled since FP16 consumes only half the memory for floating points as FP32. Doubling the batch size with FP16 ensures that GPU memory is utilized equally for both types of tests. The performance comparison of FP16 versus FP32 is shown in Figure 5 for all three frameworks used in this study. Tests were conducted on up to two PowerEdge C4140 compute nodes. Overall FP16 is 65% to 91% faster than FP32.

Although FP16 makes training faster, it requires extra work in the neural network model implementation to match the accuracy achieved with FP32. This is because some neural networks require their gradient values to be shifted into FP16 representable range, and may do some scaling and normalization to use FP16 during training. For more details, please refer to NVIDIA mixed precision training. In the future work, the image classification accuracy of applying FP16 will be compared to that of applying FP32.

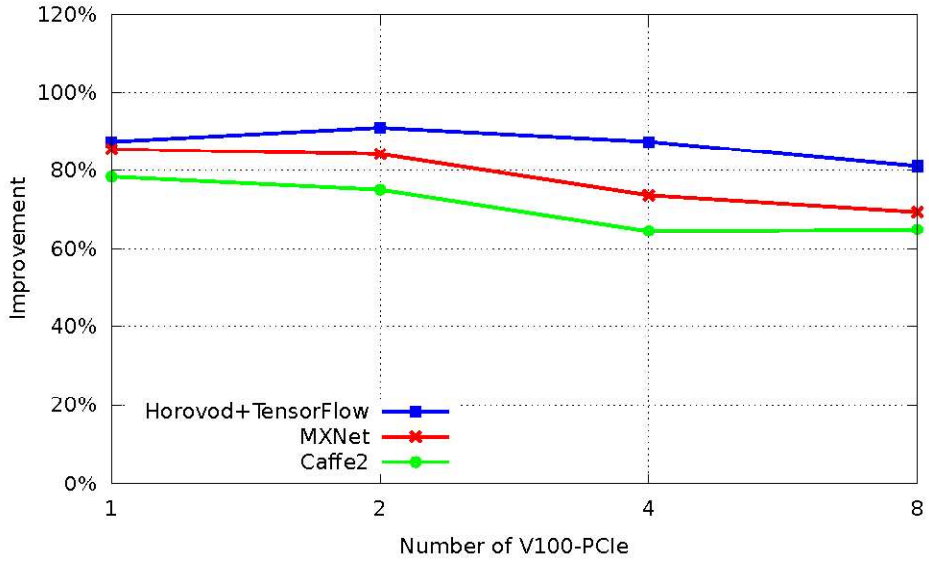


Figure 5: Performance improvement of FP16 over FP32 for Resnet50 with ILSVRC2012

3.1.2 V100 vs P100

Table 6 compares V100-SXM2 and its previous generation GPU, the P100-SXM2. To demonstrate the performance advantages of the V100 GPU over its previous generation P100 GPU, the performance of one node with four V100-SXM2 was compared to that of a node with four P100-SXM2. Figure 6 shows this performance comparison. The result shows that in FP32 mode, V100 is 26 % faster than P100 when using TensorFlow, and 52% faster with MXNet. This is because V100-SXM2 has more CUDA cores, higher clock rate and higher memory bandwidth than P100-SXM2. In FP16 mode, V100 is 103% faster than P100 with TensorFlow, and 124% faster with MXNet. The reason that V100 is much faster than P100 in FP16 mode is because P100 does not have Tensor Cores.

Table 6: Comparison between P100-SXM2 and V100-SXM2

	P100-SXM2	V100-SXM2
CUDA Cores	3584	5120

GPU Max Clock rate (MHz)	1481	1530
Tensor Cores	N/A	640
Memory Bandwidth (GB/s)	732	900
NVLink Uni-direction Bandwidth (GB/s)	80	150
Double Precision (TFLOPS)	5.1	7.5
Deep Learning (Tensor OPS)	0	120
TDP (Watts)	300	300

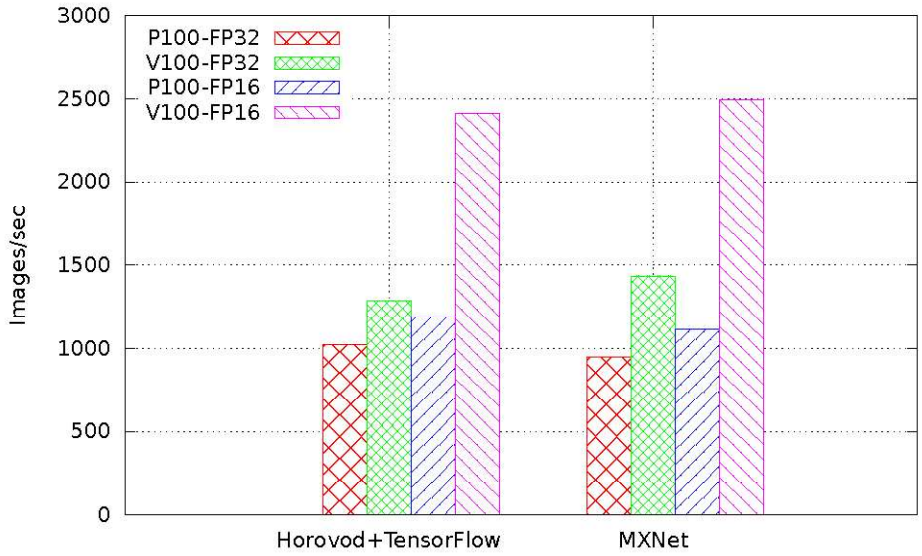


Figure 6: Performance comparison between V100-SXM2 and P100-SXM2 for Resnet50 with ILSVRC2012 within one node (four GPUs)

3.1.3 V100-SXM2 vs V100-PCIe

V100-SXM2 GPUs are recommended over V100-PCIe GPUs in the Deep Learning solution described in this document. When multi-GPU are used, V100-SXM2 has the advantage of using the faster NVLink for GPU to GPU communication over V100-PCIe which uses PCIe for GPU communication. As described in Section 2.2.1, each V100-SXM2 GPU has 6 NVLinks for bi-directional communication. The bandwidth of each NVLink is 25GB/s in uni-direction and all 4 GPUs within a node can communicate at the same time, therefore the theoretical peak bandwidth is 6*25*4=600GB/s in bi-direction. However, the theoretical peak bandwidth using PCIe is only 16*2=32GB/s as the GPUs can only communicate in order. So in theory the data communication with NVLink could be up to 600/32=18x faster than PCIe.

To evaluate the performance advantage of using NVLink over PCIe, the Peer-to-Peer (P2P, which is GPU-to-GPU within the same compute node) memory access time when running three Deep Learning frameworks was profiled using NVIDIA's command line profiler nvprof. It was found that TensorFlow implements P2P without an explicit call to the cudaMemcpyPeer () API but nvprof profiles P2P communication based on this API, therefore there is no way to profile the P2P speedup for TensorFlow with nvprof.

The P2P memory access time speedup with MXNet and Caffe2 was measured to be 3.7x and 3.2x, respectively, when using NVLink over PCIe across four GPUs in FP32 mode. However, the P2P memory accesses make up only a small portion of the whole application time, the overall application performance improvement with V100-SMX2 is a more modest 5-20% over V100-PCIe as shown in Figure 7. In Figure 7, up to four GPUs are within one node, and 8 GPUs are in two nodes. The performance improvement percentage is higher with increasing number of GPUs within one node. As we cross past a single node, the performance improvement may drop since the GPU to GPU communication in some instances has to go past the PCIe and the EDR InfiniBand link apart from the NVLINK traffic. Hence the GPU communication time overall would be higher. This performance delta between PCIe and SXM2 is expected to increase if the GPU-GPU communication in a neural network model implementation increases (Ex: model parallel instead of data parallel), and as software implementations take further advantage of the P2P architecture. Given the performance advantage of the SMX2 modules today, and the potential for further improvements with SMX2, this solution recommends the SMX2 GPUs over PCIe.

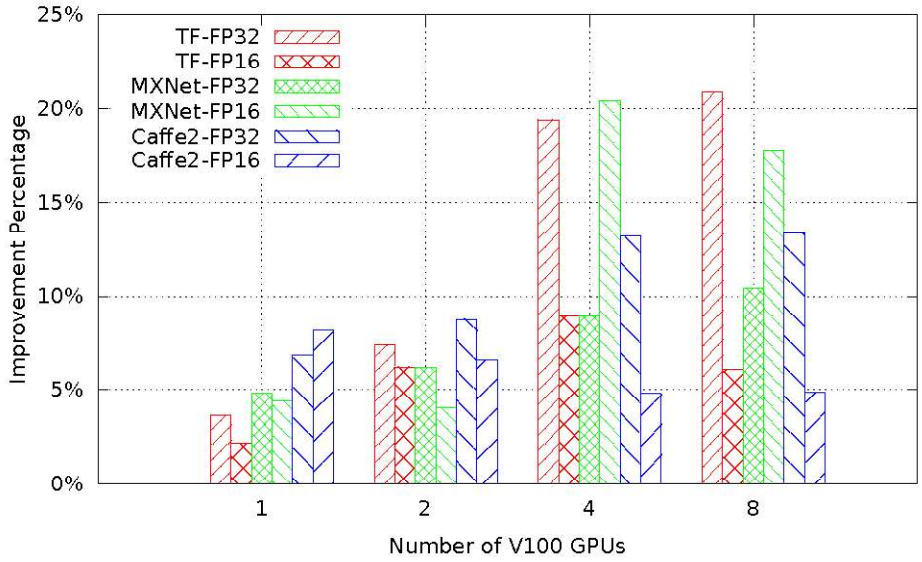
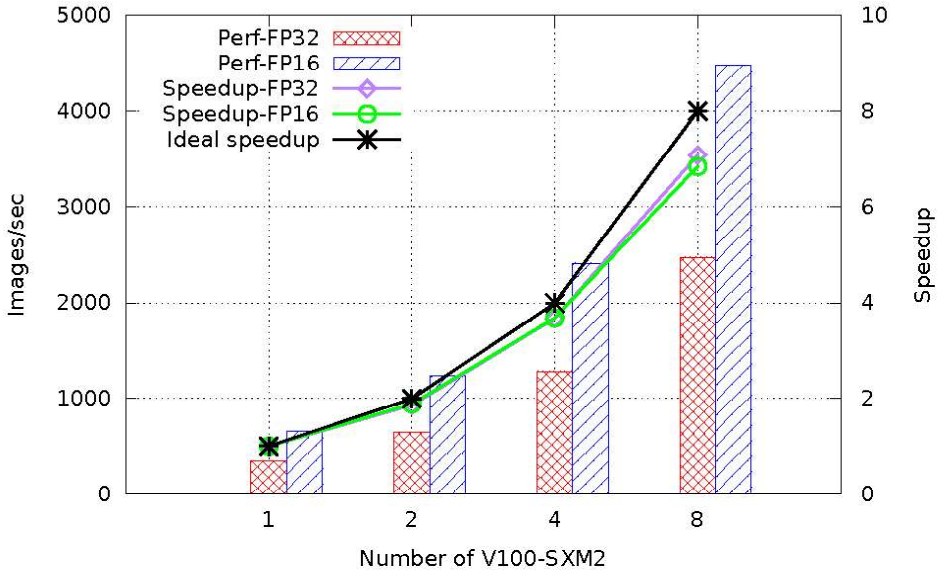


Figure 7: Performance improvement of V100-SXM2 over V100-PCIe for Resnet50 with ILSVRC2012

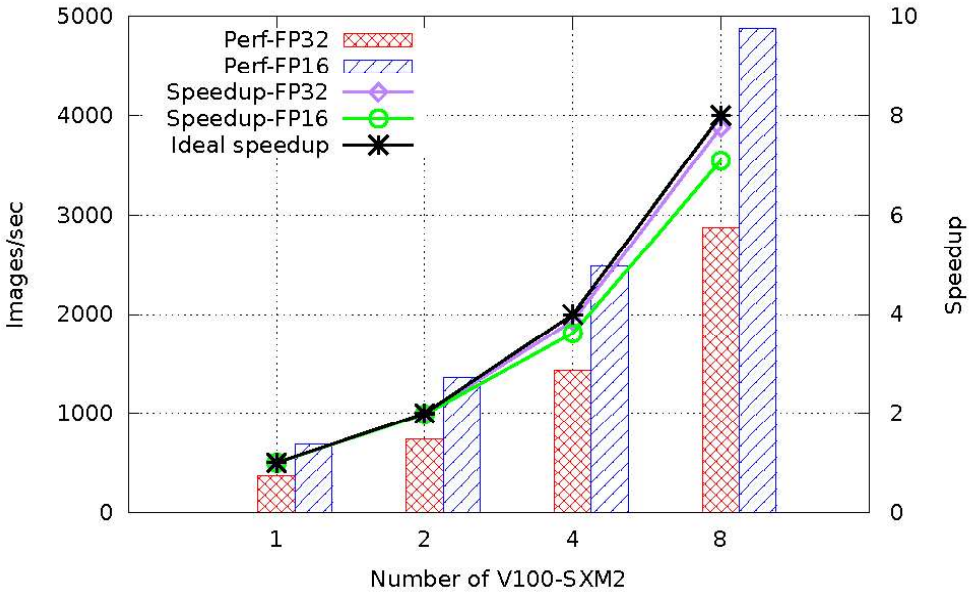
3.1.4 Scaling Performance with Multi-GPU

Figure 8 shows the scaling performance and speedup of one to eight V100-SXM2 GPUs. The multiple GPUs are either in single compute node (up to four GPUs) or across two compute nodes. Two PowerEdge C4140 nodes were used with 8 GPUs in total.

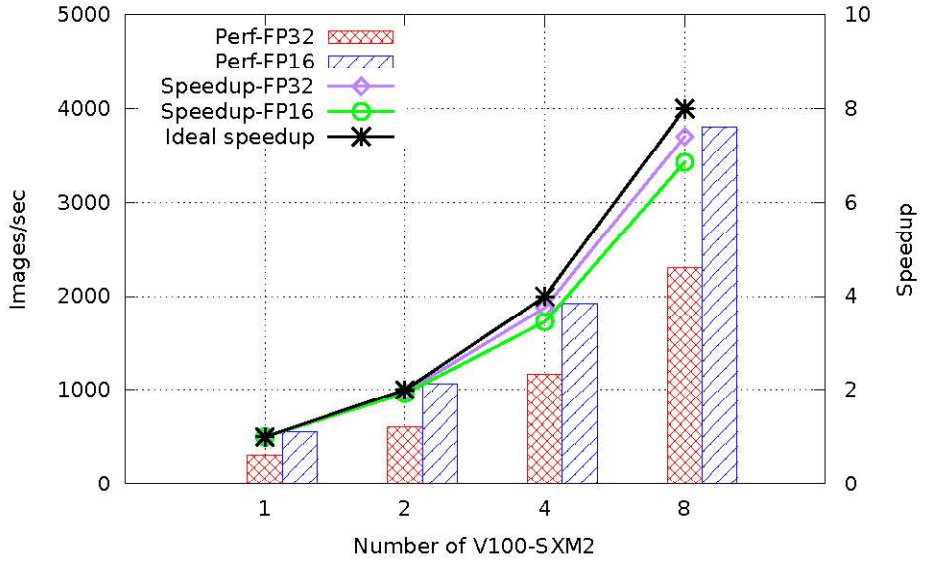
As discussed in Section 3.1.1, there are two popular modes in Deep Learning training, FP32 and FP16, and both modes were examined in this test. The time consuming part of the training phase are the matrix operations. In FP32 mode, all floating point numbers use the standard 32-bit representation. In FP16 mode, however, some floating point numbers are represented with only 16-bit. It can be seen that, across the three frameworks and when using 8 GPUs, MXNet scales the best with 7.8x speedup in FP32 mode and 7.1x speedup in FP16 mode. The speedup of TensorFlow is 7.1x in FP32 and 6.8x in FP16, respectively. For Caffe2, the speedup is 7.4x in FP32 and 6.9x in FP16, respectively. The results indicate that the Deep Learning training performance scales well across more than one compute node with this solution. The next test examines a multi-node solution.



(a) Horovod+TensorFlow across 8 GPUs using the ILSVRC2012 dataset with Resnet50



(b) MXNet across 8 GPUs using the ILSVRC2012 dataset with Resnet50

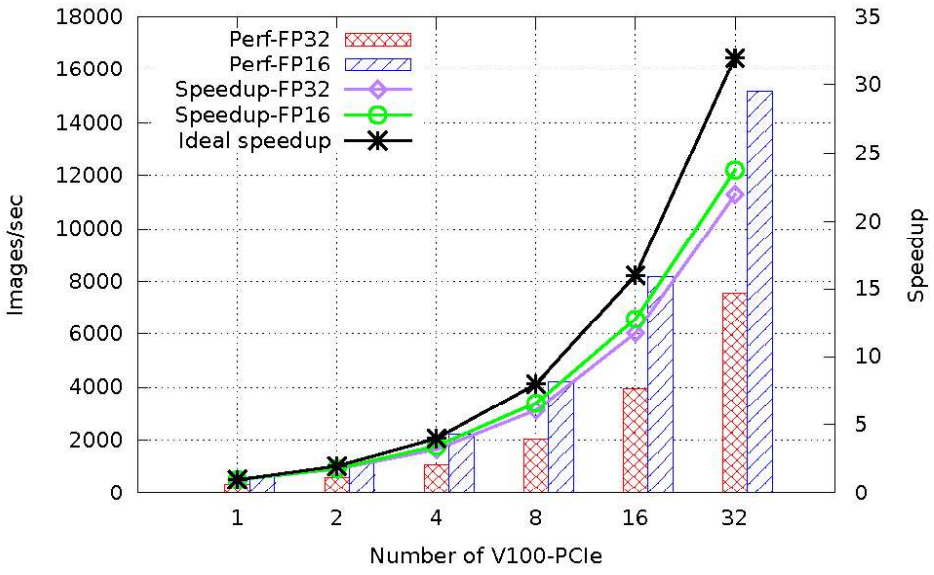


(c) Caffe2 across 8 GPUs using the ILSVRC2012 dataset with Resnet50

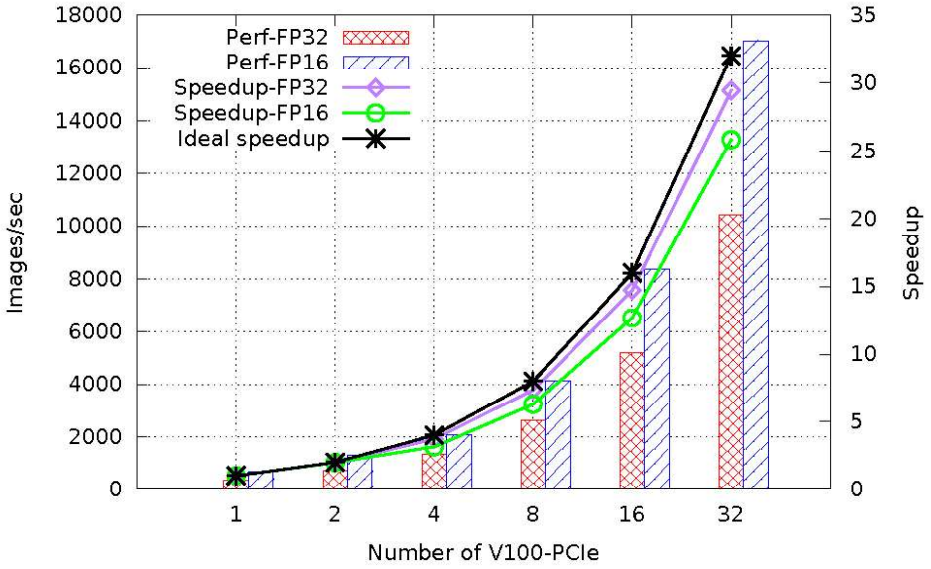
Figure 8: The scaling performance of Deep Learning training on V100-SXM2

To demonstrate the scalability with more than two compute nodes, the same Deep Learning training benchmarks were executed on a solution with eight PowerEdge C4140 nodes with four V100-PCIe GPUs per server. The multi-node test bed available at the time of writing provided PCIe based GPUs. Given that the performance difference between PCIe and SMX2 GPUs is well understood (between 5-20% as presented in Section 3.1.3), the PCIe GPUs were considered a reasonable test to understand the scalability of the frameworks. It is expected that the SMX2 GPUs will demonstrate similar scalability patterns.

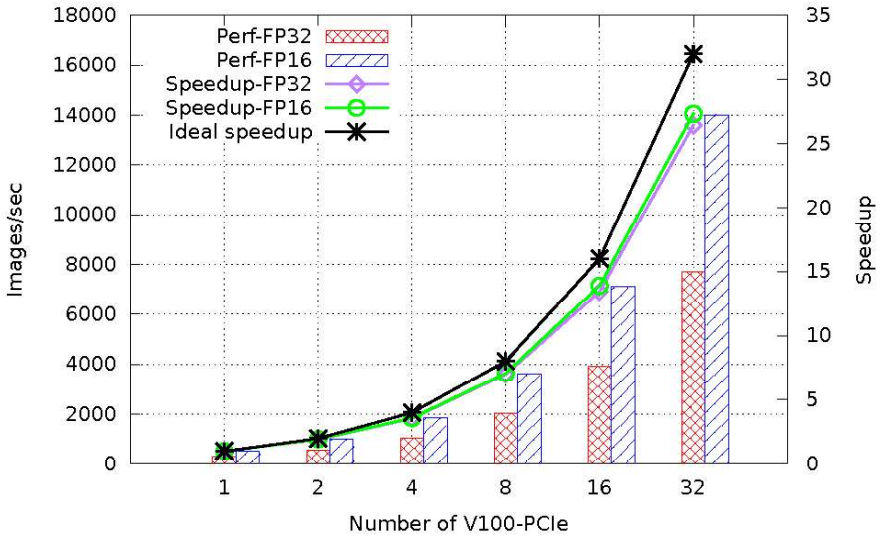
The results are presented in Figure 9. Since eight compute node are used, there are 32 GPUs in total. It can be seen that among the three frameworks, with two nodes and eight GPUs, the speedup in all three frameworks are similar to the speedup with V100-SXM2 GPUs (presented in Figure 8). When using 32 GPUs, MXNet scales the best with 29.4x speedup in FP32 mode and 25.8x in FP16 mode. TensorFlow also scales well with 22.0x speedup in FP32 mode and 23.7x in FP16 mode. For Caffe2, the speedup is 26.5x in FP32 and 27.3x in FP16.



(a) Horovod+TensorFlow across 32 GPUs using the ILSVRC2012 dataset with Resnet50



(b) MXNet across 32 GPUs using the ILSVRC2012 dataset with Resnet50



(c) Caffe2 across 32 GPUs for Resnet50 using the ILSVRC2012 dataset

Figure 9: The scaling performance of Deep Learning training on V100-PCIe

3.1.5 Storage Performance

The impact of different storage sub-system options for the Deep Learning Solution was evaluated next. Experiments were designed to measure the performance of three different convolutional neural networks on four types of storage systems. The TensorFlow framework was used and the three neural networks used were AlexNet, ResNet50, and VGG16. The batch size used for benchmarking was 128 for VGG16, and 256 for both AlexNet and ResNet50. All experiments were running in FP16 mode to stress the I/O operations in neural network training. The four systems are described below.

- Isilon F800

- NFS on Near-Line SAS drives hosted by the head node and exported over NFS via IPoIB
- NFS on SSD drives hosted by the head node and exported over NFS via IPoIB
- NVMe

The Isilon F800 evaluated in this experiment has the same configuration as described in Section 2.5. Before running any benchmark on Isilon, the command "`isi_for_array isi_flush --l1 --l2 --lk`" was used to clear the cache on all four Isilon F800 nodes.

The NFS file share hosted on Near-Line SAS drives local to the head node (NL SAS NFS) consists of 12 TB NL SAS drives on the head node in a RAID 50 volume, formatted as an XFS file system and exported via NFS to the compute nodes over IPoIB.

The third storage system that was evaluated includes an NFS file share hosted on SATA SSDs local to the head node and exported to the compute nodes over IPoIB. This option is called "SSD NFS" as shorthand. This configuration was included in the experiment to evaluate the performance an SSD based storage solution. Four SATA SSD disks local to the head node were configured in a RAID 0 volume since the goal was to test the maximum performance. The SSDs were 1.92TB read-intensive drives. Since most disk operation in Deep Learning training are read-intensive, 1 Drive Writes Per Day (DWPD) drives were selected. The RAID 0 volume was formatted as an XFS file system and exported to the compute nodes via IPoIB. The goal of this configuration was to understand the benefit of SSD drives to the read operations in Deep Learning training and to determine if this could be a viable option for a storage solution or as a scratch space for temporary files. For the purpose of this experiment, an 8TB SSD solution was deemed sufficient. For environments that chose such an SSD solution for a production NFS system, RAID 6 or RAID 50 is recommended to protect data against disk failures. The quantity and capacity of the SSDs should also be increased to accommodate user and project storage needs.

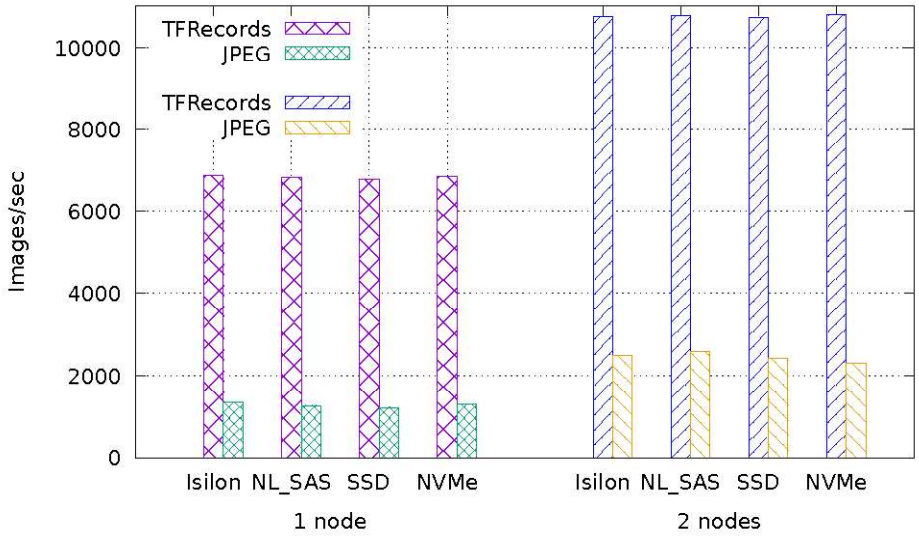
The final storage subsystem evaluated used NVMe drives. The compute nodes have an option for local NVMe devices in addition to SAS or SATA, and these devices can be used as local scratch space. In a typical Deep Learning training cycle, the same dataset is used multiple times. If the dataset is large and cannot fit in the compute node system memory, a larger capacity local NVMe drive can be used to provide much faster disk-to-memory I/O than SSD drives. The PowerEdge C4140 can support up to two NVMe devices in PCI-e card form factor in the rear PCIe slots in the server chassis. 1x 1.6 TB NVMe device was tested in this study as the other PCIe slot was populated by the Mellanox InfiniBand EDR adapter. The random read performance of the single NVMe device is 1080,000 IOPS and the sequential read performance is 6,400 MB/s. This option is called "NVMe" in the results below.

Since the entire ILSVRC2012 benchmark dataset is only ~140 GB and can easily fit into system memory, the size of the dataset was increased 10x by applying 10 different data augmentation techniques to each JPEG image in the dataset. The augmented images were then converted to a TensorFlow TFRecords database. TFRecords file format is a binary format that combines multiple raw image files together with their metadata information into one binary file. It maintains the image compression offered by the JPEG format and the total size of the dataset remained the same. Training performance was measured with the TensorFlow framework using both TFRecords database as well as the raw augmented JPEG images, and the results are presented in Figure 10. The program did not run to completion with Resnet50 and JPEG images for all storage options, so there is no corresponding values in Figure 10 (b). This issue will be studied as part of future work.

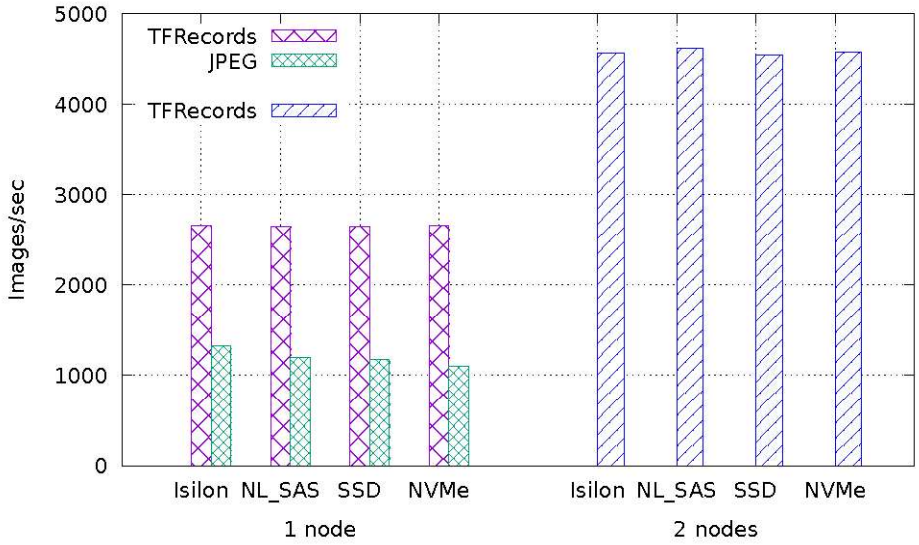
The performance studies in the Figure 10 below have been conducted with the 1 and 2 nodes of the PowerEdge C4140 with V100 SXM2 GPUs interconnected with EDR InfiniBand as listed in Table 5.

The results show that for each of the three neural networks, there is no perceptible difference with respect to the type of storage subsystem that was used. The conclusion is that the frameworks tested do not significantly

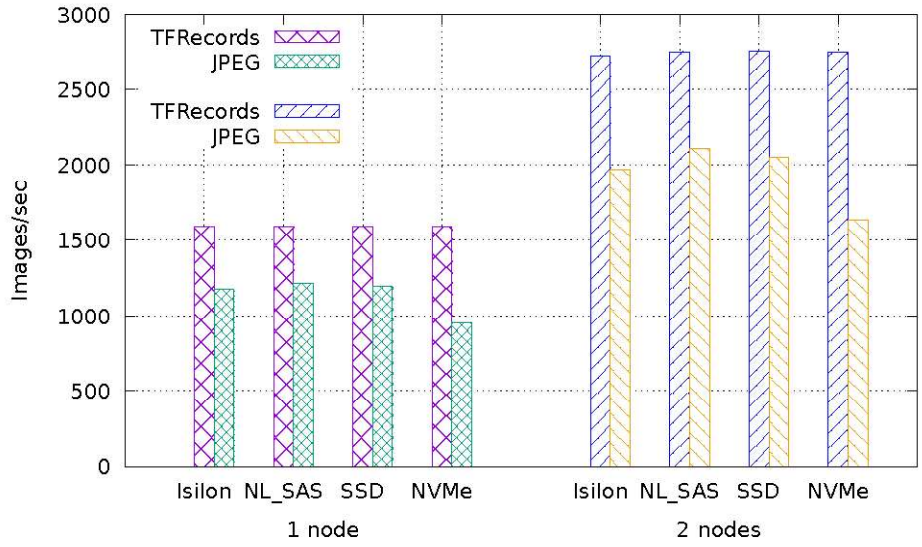
stress the I/O to be able to demonstrate notable difference in performance between the storage options. But, it turns out that the performance with the TFRecrods database is much better than using raw JPEG images. The performance delta between these two different image formats is much larger in the less complicated neural networks like AlexNet. For instance, the performance advantage when using TFRecords over raw JPEG images is ~23%-40% for VGG16, but around 5x (500%!) for AlexNet. This is because the TFRecords format packs many raw JPEG images together making the data access more efficient, and additionally, TensorFlow includes input pipeline optimization for TFRecords format but no corresponding optimization API for raw images. Because of this reason, the performance variation with JPEG images in different storages is quite large, especially in VGG16. Profiling and further exploration is ongoing to understand the variation with JPEG images. The recommended format to use would be TFRecords.



(a) AlexNet



(b) Resnet50



(c) VGG16

Figure 10: Neural network training performance with different storages systems and image database options. The batch size is 256, 256 and 128 for AlexNet, Resnet50 and VGG16, respectively. All training are in FP16 mode.

The training performance of the tested neural networks are affected by different flags in the benchmark. It was found that the training performance of Resnet50 could be improved by changing the default value of the flag "--datasets_num_private_threads" to four. There was no obvious performance improvement for VGG16 and AlexNet. The performance comparison between the baseline version and the tuned version for Resnet50 is shown in Figure 11. The baseline performance is the performance presented in Figure 10(b). With one node, the performance was improved from 2,657 images/sec to 2,940 images/sec. With two nodes, the performance was improved from 4,560 images/sec to 5,590 images/sec.

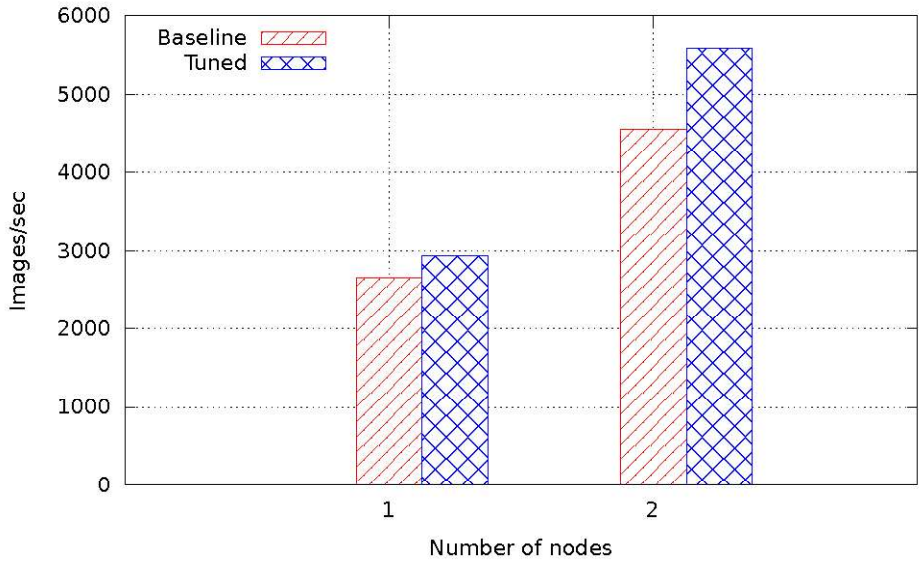


Figure 11: The performance comparison of Resnet50 between the baseline and tuned versions

Figure 12: The profiled disk throughput with InsightIQ when running Resnet50 with Isilon

To better understand the underlying storage system, the Isilon storage I/O performance was profiled using Isilon InsightIQ while training the model. The Isilon InsightIQ was described in 2.5. Only the Isilon storage with TFRecords dataset was profiled since all storage systems displayed similar performance and the lessons from one profiling exercise should be broadly applicable for this use case. Figure 12 shows an example of InsightIQ snapshot of the disk throughput when running Resnet50. The disk throughput in the figure is decreasing because more data were cached in Isilon memory. The full disk profiling data for AlexNet, Resnet50, and VGG16 are shown in Table 7. The training performances are also added in the table for validation. Take Resnet50 for example, the training performance is 2,940 images/sec. The average size of each image is 0.113 MB (the total training images size is 1,448,283,629,516 bytes, divided by 12811670 image files), so the expected disk throughput is 2,940 images/sec * 0.113 MB * 8 b/B = 2,658 Mb/s which matches the actual disk throughput 2,680 Mb/s. The same conclusion also applies to AlexNet and VGG16 neural networks.

Table 7: The disk metrics in Isilon F800 with single compute node

	AlexNet	Resnet50	VGG16
Average Disk Operation Size (MB)	8.61	9.17	8.81
Disk Read IOPS (K/s)	344	212	111
Disk Throughput (Mb/s)	6370	2680	1440
Training Performance (Images/sec)	7095	2940	1590

Figure 13 illustrates the CPU utilization, memory usage and GPU utilization on one compute node, and the network and disk throughput on Isilon F800 when running Resnet50 FP16 training with 10 times ILSVRC2012 TFRecords dataset. The training speed keeps 2,940 images/sec throughout the whole training time.

The CPU utilization is around 33% out of 40 cores, which are used for I/O, TFRecords parsing and JPEG decoding. This verifies that the CPUs are not the bottleneck in Deep Learning training of Resnet50. For memory, only around 19% of 384GB memory are used. The buffer usage is increasing to cache the data in memory until the whole memory is used. But the whole dataset cannot be all fitted into memory. This is because 10x ILSVRC2012 dataset was used by intention, the total size of the dataset is around 1.35TB which cannot be fully

cached into the system memory on one compute node. The cache was not cleared after the benchmark stopped, that is why the used memory is closed to zero but cache still does not decrease. Although the memory cannot cache the whole dataset, the network and file system are fast enough to feed data to GPUs to keep them stay at high utilization and to keep the training speed at 2,940 images/sec. The average GPU utilization was around 380% across four GPUs. This high utilization indicates that other parts (CPU, memory, network, etc.) of one compute node are not the performance bottleneck.

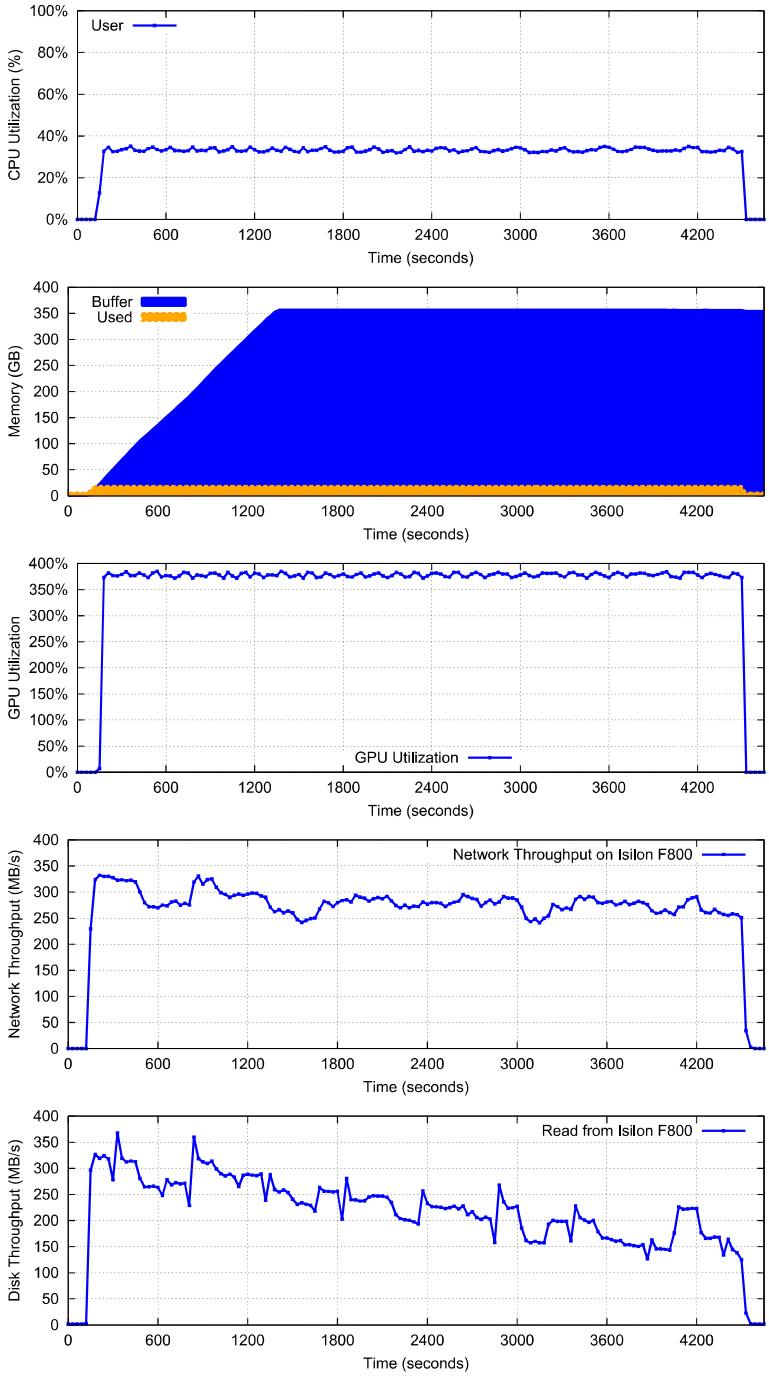
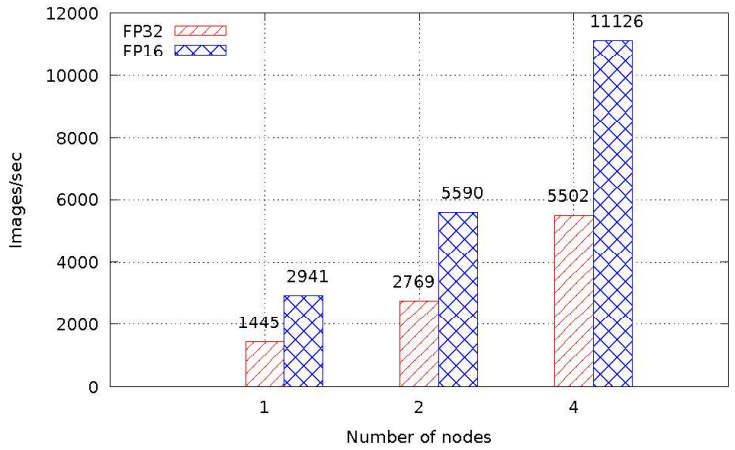


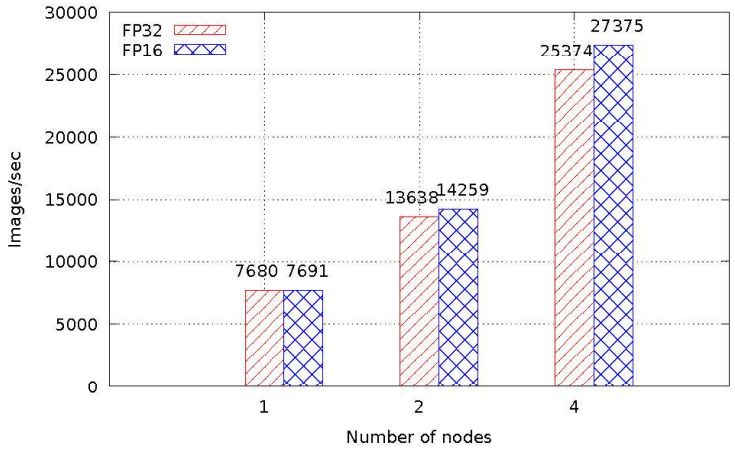
Figure 13: The CPU utilization, memory usage and GPU utilization on one compute node, and network and disk throughput on Isilon when running Resnet50 with 10x ILSVRC2012 TFRecords dataset

On Isilon F800, the network throughput is around 300 MB/s to maintain high GPU utilization. This is the data transfer throughput of Resnet50 training and it almost does not change during the whole training. The network bandwidth is 56 Gb/s (7 GB/s) between the IB EDR Switch and the FDR-40 GigE Gateway as shown in Figure 2. Two more InfiniBand network connections can be added (three connections in total) to match the bandwidth of four Isilon nodes, which is 4 x 40 Gb/s. This supports up to 70 ((7 GB/s)*3/(300 MB/s)) compute nodes. This number is far more than the total number of 36 ports in IB EDR Switch. This leaves more room for scaling the cluster with more compute nodes.

When no data cached in Isilon F800, the disk throughput is the same as network throughput at the beginning of training. As more data are cached into Isilon memory, the disk throughput is decreasing. Since the dataset is too large to fit into system memory on compute and Isilon node, the system has to fetch the data from SSDs on Isilon. That is the reason of consistent disk read throughout Resnet50 training.



(a) Resnet50



(b) AlexNet

Figure 14: Scaling results with Isilon F800. In FP32 mode, the batch size is 128 and 512 for Resnet50 and AlexNet, respectively. In FP16 mode, the batch size is 256 and 1024 for Resnet50 and AlexNet, respectively.

Figure 14 summaries the scaling results on up to four compute nodes (16 GPUs) for Resnet50 and AlexNet with Isilon F800. To stress the I/O, the batch size for AlexNet is changed to 512 in FP32 mode and 1024 in FP16 mode.

There are ongoing studies to further stress the storage subsystems with other models (model parallelism like seq2seq) and datasets to understand the performance implications of deep learning workloads at scale.

3.2 Deep Learning Inference

Inference is the end goal of Deep Learning. The inference performance tends to be either latency-focused or throughput-focused. On one hand, latency-focused scenarios are time sensitive (e.g. face recognition), with time to solution taking priority over efficient hardware utilization. In such case, the batch size can be as small as one. On the other hand, in cases where delayed batch processing is acceptable large batch sizes can be used to increase throughput.

This section quantifies the performance of Deep Learning inference using NVIDIA's TensorRT library. TensorRT, previously called GIE (GPU Inference Engine), is a Deep Learning inference engine for production deployment of Deep Learning applications which aims to maximize inference throughput and efficiency. TensorRT provides users the ability to take advantage of fast reduced precision instructions provided in the P100 and V100 GPUs.

In the experiments conducted in this study, TensorRT 4.0.0.3 was used. As described in Section 3.1.1, deep learning researchers have found they are able to achieve the same inference accuracy with FP16 (16-bit floating point) as can be had with FP32. Many applications require only INT8 (8-bit integer) or lower precision to keep an acceptable inference accuracy. TensorRT included support for INT8 operations in version 2 of the software. All inference experiments were performed on a single V100 GPU. Multi-GPU results were not included since most inference jobs are large embarrassingly parallel batch jobs and there is not much communication between the GPU. Therefore for most use cases linear scalability can be expected when using multi-GPUs for inferencing.

Figure 15 shows the inference performance with TensorRT on Resnet50 model with different batch sizes. At the time of writing, there is a known issue on V100 GPUs where running a model with INT8 works only if the batch size is evenly divisible by 4, so there is no resultant performance value when the batch size is 1. The results show that the INT8 mode is 2.5x – 3.5x faster than using FP32 when the batch size is less than 64, and ~3.7x faster when the batch size is greater than 64. This is expected since the theoretical speedup of INT8 is 4x compared to FP32 if only multiplications are performed and no other overhead is incurred. However, there are kernel launches, occupancy limits, data movement and mathematical operations other than multiplications, so the speedup is reduced to about 3x faster.

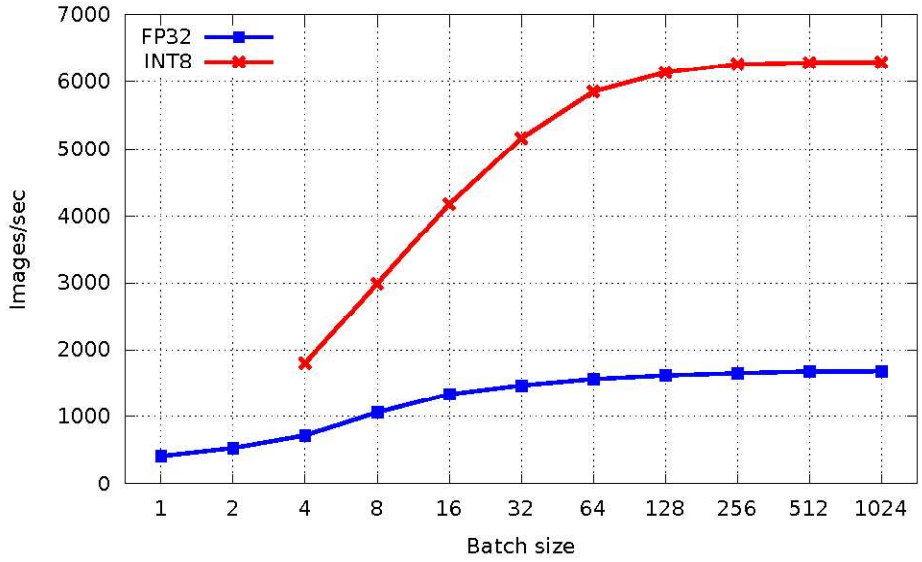


Figure 15: Inference performance with INT8 vs FP32 for Resnet50 model

Figure 15 also illustrates the performance difference when using different batch sizes. It can be seen that without batch processing the inference throughput is much lower. This is because the GPU is not assigned enough work in each iteration to keep it busy. The larger the batch size, the higher the inference throughput, although this advantage begins to flatten as batch size increases. The largest batch size is limited by GPU memory.

The accuracy of using INT8 versus FP32 was compared to confirm that the inference performance gains with INT8 do not incur a penalty with inaccurate results. To ensure INT8 data encodes the same information as FP32 data, a calibration method is applied in TensorRT to convert FP32 to INT8 in a way that minimizes the loss of information. More details of this calibration method can be found in the presentation 8-bit Inference with TensorRT from the GTC 2017 conference. The ILSVRC2012 validation dataset was used for both calibration and benchmarking. The validation dataset has 50,000 images and was divided into batches where each batch has 25 images. The first 50 batches were used for calibration purpose and the rest of the images were used for accuracy measurement. Several pre-trained neural network models were used in our experiments, including ResNet-50, ResNet-101, ResNet-152 , VGG-16, VGG-19 , GoogLeNet and AlexNet. Both top-1 and top-5 image classification accuracy were recorded using FP32 and INT8 and the accuracy difference between FP32 and INT8 was calculated. The top-1 accuracy means the percentage of the total number of matches from the validation set where the first predicted output (highest confidence result) from the model is the same as the ground truth. The top-5 accuracy means the percentage of the total matches from the validation set where the ground truth falls in the first five of the highest confidence predictions by the model. The result is shown in Table 8. It is clear from the results that the accuracy difference between FP32 and INT8 is not significant, between 0.02% - 0.18% for all test cases. This means very little accuracy is lost with INT8, while achieving a 3x speed up over FP32.

Table 8: The accuracy comparison between FP32 and INT8

Neural Network Model	FP32		INT8		Difference	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
ResNet-50	72.90%	91.14%	72.84%	91.08%	0.07%	0.06%
ResNet-101	74.33%	91.95%	74.31%	91.88%	0.02%	0.07%

ResNet-152	74.90%	92.21%	74.84%	92.16%	0.06%	0.05%
VGG-16	68.35%	88.45%	68.30%	88.42%	0.05%	0.03%
VGG-19	68.47%	88.46%	68.38%	88.42%	0.09%	0.03%
GoogLeNet	68.95%	89.12%	68.77%	89.00%	0.18%	0.12%
AlexNet	56.82%	79.99%	56.79%	79.94%	0.03%	0.06%

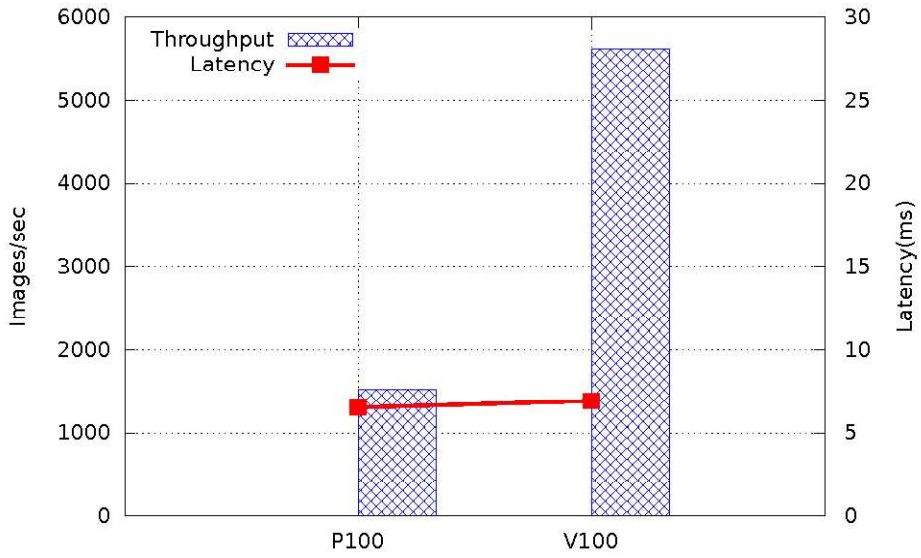


Figure 16: Resnet50 inference performance on V100 vs P100 (V100 is 3.7x faster than P100)

To demonstrate the inference advantage of V100 over P100, tests in FP16 mode were performed on both V100 and P100 and the result is shown in Figure 16. Batch size of 39 was used for V100 and 10 for P100. Different batch sizes were chosen for the two GPU generations such that the inference latencies were almost identical (~7ms in the figure). The result shows that when inference latency is held constant, the inference throughput of the V100 is 3.7x faster when compared to the P100.

3.3 NVIDIA DIGITS Tool and the Deep Learning Solution

DIGITS is a web frontend to Caffe, Torch and TensorFlow, developed by NVIDIA. The user can use the DIGITS graphical user interface for Deep Learning training and inference. The Deep Learning Solution presented in this paper integrates the DIGITS software with the cluster software making it trivial for the system administrator to deploy DIGITS for the users, and easy for the user to use DIGITS.

In order to use DIGITS, the user first needs to allocate the resource, load the DIGITS module and start DIGITS service by running "`digits-devserver`". These steps are listed in Figure 17.

```
[user1@headnode ~]$ srun -N 1 -n 8 -p shareq --gres=gpu:2 --mem=96G -t 8:00:00 --pty bash
[user1@node001 ~]$ module load shared digits/6.1.1
[user1@node001 ~]$ mkdir -p /home/user1/digits-logs
[user1@node001 ~]$ export DIGITS_JOBS_DIR=/home/user1/digits-logs
[user1@node001 ~]$ /cm/shared/apps/digits/6.1.1/digits-devserver
```

Figure 17: How to initialize DIGITS on the Deep Learning Solution

Once the DIGITS server is up and running, a web browser can be used to navigate to the home page of DIGITS. As shown in Figure 18, the DIGITS home page can be accessed from “http://node001:5000”. To know how to use DIGITS in more details, refer to Bright Computing’s Machine Learning Manual Chapter 2 which includes an example on handwritten digits recognition using a Caffe backend.

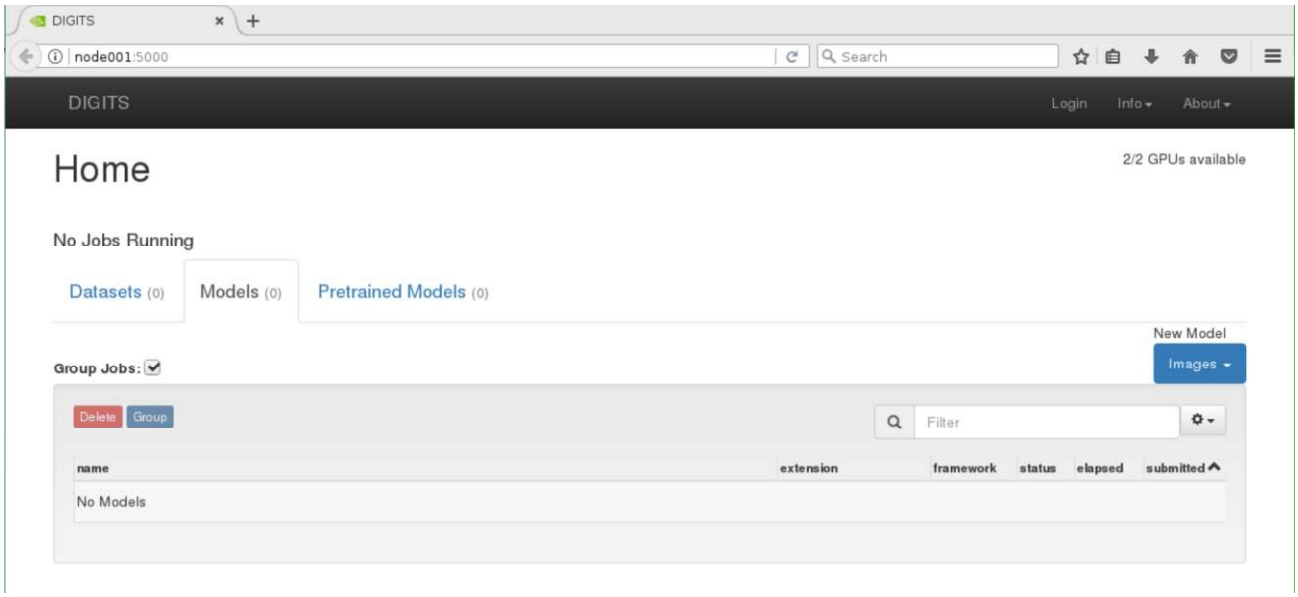


Figure 18: DIGITS home page

4 Containers for Deep Learning

Deep Learning frameworks tend to be complex to install and build with a myriad of library dependencies. These frameworks and their requisite libraries are under constant development, which makes test environment and test result reproducibility a challenge for researchers. Another layer of complexity is that most enterprise data centers use Red Hat Enterprise Linux (or its derivatives) whereas Ubuntu is the default target for most Deep Learning frameworks.

Containerization technology has surged in popularity as it is a powerful tool for handling the three issues just described – portable test environment, reduced dependency on the underlying operating system and better test result reproducibility. A container packs all the environment and libraries an application needs into an image file and this container can be deployed without any additional changes. It also allows users to easily create, distribute and destroy a container image. Compared to Virtual Machines, containers are lightweight with less overhead. In this study, Singularity, a container designed specifically for use in HPC environments, is used to containerize different Deep Learning frameworks. The results presented in this section demonstrate that the containerized version can achieve the same performance as a bare-metal install while simplifying the build and deployment of Deep Learning frameworks.

4.1 Singularity Containers

Singularity is developed by Lawrence Berkeley National Laboratory to provide container technology specifically for HPC. It enables applications to be encapsulated in an isolated virtual environment to simplify application deployment. Unlike virtual machines, the container does not have a virtual hardware layer and its own Linux kernel inside the host operating system (OS), therefore the overhead and the performance loss are minimal. The main goal of the container is reproducibility. The container has all environment and libraries an application needs to run, and it is portable so that other users can reproduce the results the container creator generated for that application. To use Singularity container, the user only needs to load its module using the command "`module load singularity`" inside a Slurm script which will be described in Section 5.3.

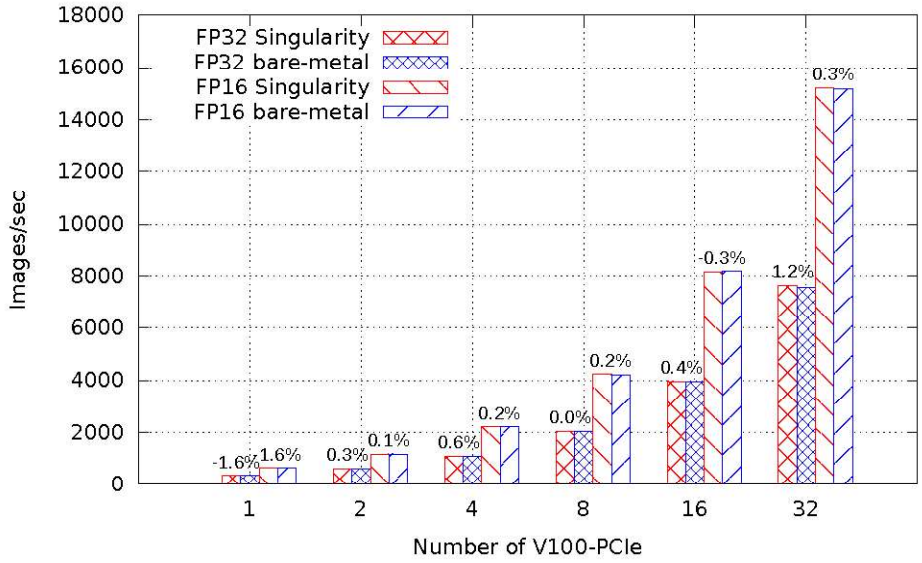
Many HPC applications, especially Deep Learning applications, have extensive library dependencies and it is time consuming to solve these dependencies and debug build issues. Most Deep Learning frameworks are developed in Ubuntu but they need to be deployed to Red Hat Enterprise Linux (RHEL). It is therefore beneficial to build those applications once in a container and then deploy them anywhere. The most important goal of Singularity is portability which means once a Singularity container is created, the container should be able to run on any system. However, there may be kernel dependencies to consider if a user needs to leverage any kernel specific functionality (e.g. OFED). Usually a user would build a container on a laptop or a server, a cluster or a cloud, and then deploy that container on a server, a cluster or a cloud.

When building a container, one challenge is when using GPU-based systems. If GPU drivers are installed inside the container, and the driver version does not match the host GPU driver, then an error will occur. Hence the container should always use the host GPU driver. The next option is to bind the paths of the GPU driver binary file and libraries to the container so that these paths are visible to the container. However, if the container OS is different than the host OS, such binding may have problems. For instance, assume the container OS is Ubuntu while the host OS is RHEL, and on the host the GPU driver binaries are installed in `/usr/bin` and the driver libraries are installed in `/usr/lib64`. Note that the container OS also has `/usr/bin` and `/usr/lib64`; therefore, if we bind those paths from the host to the container, the other binaries and libraries inside the container may not work anymore because they may not be compatible across different Linux distributions. One workaround is to move all those driver related files to a new central directory location that does not exist in the container and then bind that central location.

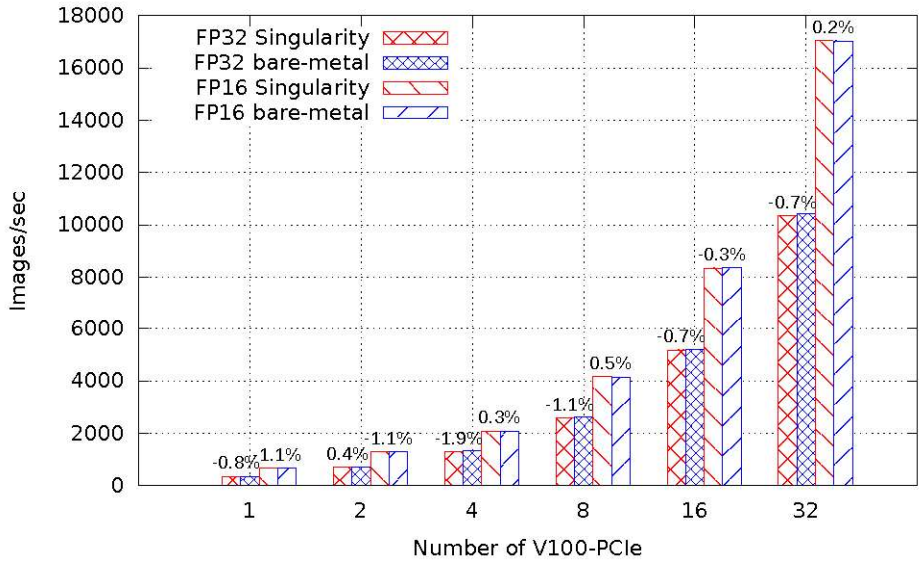
The second solution is to implement the above workaround inside the container so that the container can use the driver related files automatically. This feature has already been implemented in the development branch of Singularity repository. A user simply needs to use the option "--nv" when launching the container. However, a cluster environment typically installs GPU driver in a shared file system instead of the default local path on all nodes, and in this case, Singularity is unable to find the GPU driver path since the driver is not installed in the default or common paths (e.g. `/usr/bin`, `/usr/sbin`, `/bin`, etc.). Even if the container is able to find the GPU driver and the corresponding driver libraries and the container is built successfully, the host driver version on the target system must be updated enough to support the GPU libraries which were linked to the application when building the container, else an error will occur due to outdated and incompatible versions between the host system and the container. Given the backward compatibility of GPU drivers, the burden is on the cluster system administrators to keep GPU drivers up to date to ensure the cluster GPU libraries are equal to or newer than the versions of the GPU libraries used when building the container.

Another challenge is when using InfiniBand with the containers because the InfiniBand driver is kernel dependent. There should be no issues if the container OS and host OS are similar or compatible. For instance, RHEL and Centos are compatible, and Debian and Ubuntu are compatible. But if these two OSs are not compatible, then there will be library compatibility issues if the container attempts to use the host InfiniBand driver and libraries. If the InfiniBand driver is installed inside the container, then the drivers in the container and the host might not be compatible since the InfiniBand driver is kernel dependent and the container and the host share the same kernel. If the container and host have different InfiniBand drivers, then a conflict will occur. The Singularity community is working to solve this InfiniBand issue. The current solution is to ensure the container OS and host OS are compatible and allow the container to reuse the InfiniBand driver and libraries on the host. These are only workarounds. The container community is still pushing hard to make containers portable with ease across platforms.

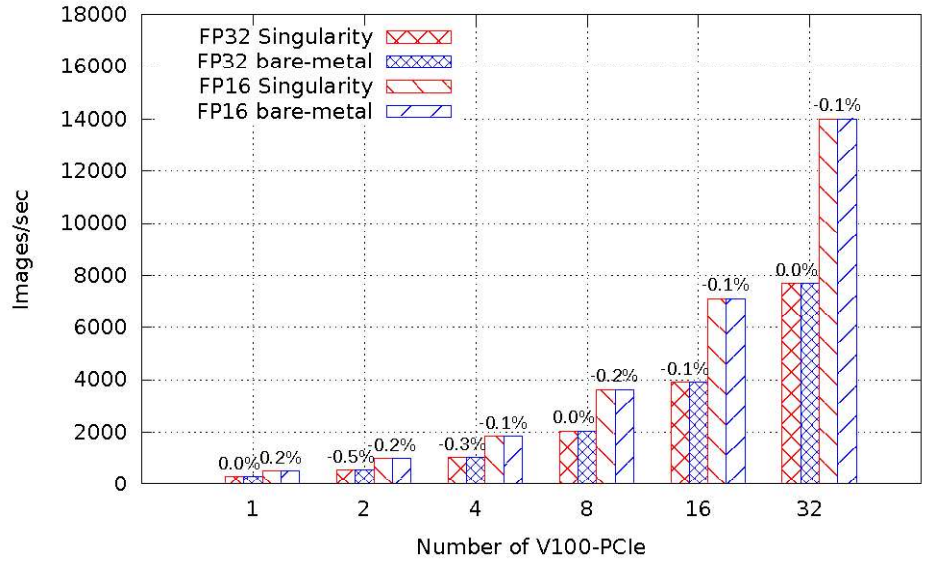
Figure 19 shows the performance comparison between runs inside Singularity and directly on bare-metal for Horovod+TensorFlow, MXNet and Caffe2 frameworks. All the hardware and software used in this benchmarking are the same as used in Section 3.1.4 and described in Table 5. The percentage number in the figure denotes the percentage difference between performance using Singularity and bare-metal ((singularity performance – bare metal performance) / bare metal performance). It can be seen that the maximum performance difference between Singularity and bare-metal is within 1.9% which is in a reasonable run-to-run variation range.



(a) Horovod+TensorFlow



(b) MXNet



(c) Caffe2

Figure 19: Singularity container vs bare-metal for Resnet50 using ILSVRC2012 dataset

4.2 Running NVIDIA GPU Cloud with the Ready Solutions for AI - Deep Learning

NVIDIA GPU Cloud (NGC) is a cloud that hosts the Docker container images of many Deep Learning frameworks and HPC applications. The Deep Learning frameworks include Caffe, Caffe2, CNTK, TensorFlow, MXNet, Theano, and so on. These frameworks are optimized for NVIDIA GPUs.

Docker is a very prominent containerization technology used avidly in the context of deep learning frameworks. There are several pros and cons of Docker and a comparison between Docker and Singularity containers is shown "Singularity Containers for HPC & Deep Learning". NVIDIA added the GPU support in Docker.

Docker is not installed by default in this solution. However, the administrator is expected to simply run `cm-docker-setup` utility provided by Bright Cluster Manager. After installing Docker, the user is able to use the Docker images provide by NVIDIA GPU Cloud which will be discussed next.

To use NGC, a user needs to first register in the cloud and get an API key that is specific to that user. Figure 20 shows the registration page. The user can create an account based on the instructions on the screen.

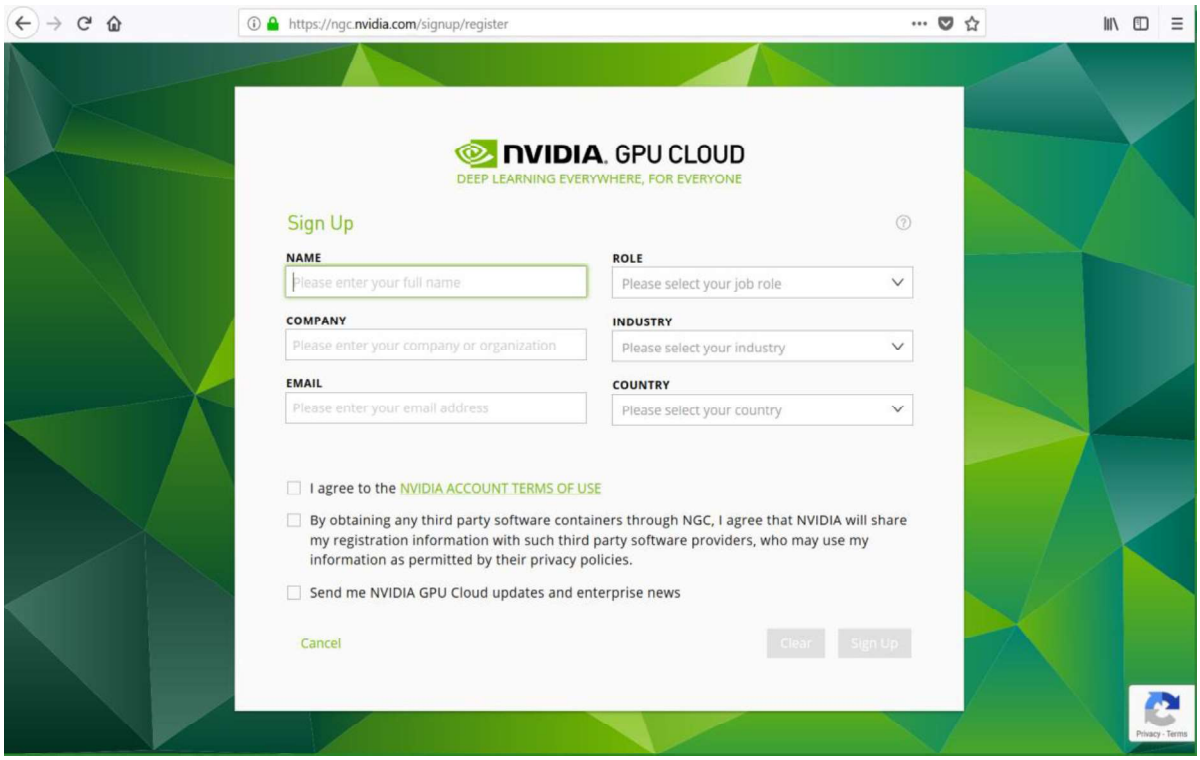


Figure 20: NVIDIA GPU Cloud register page

After creating the account, the user will be forwarded to the registry page which lists the available repositories (Figure 21).

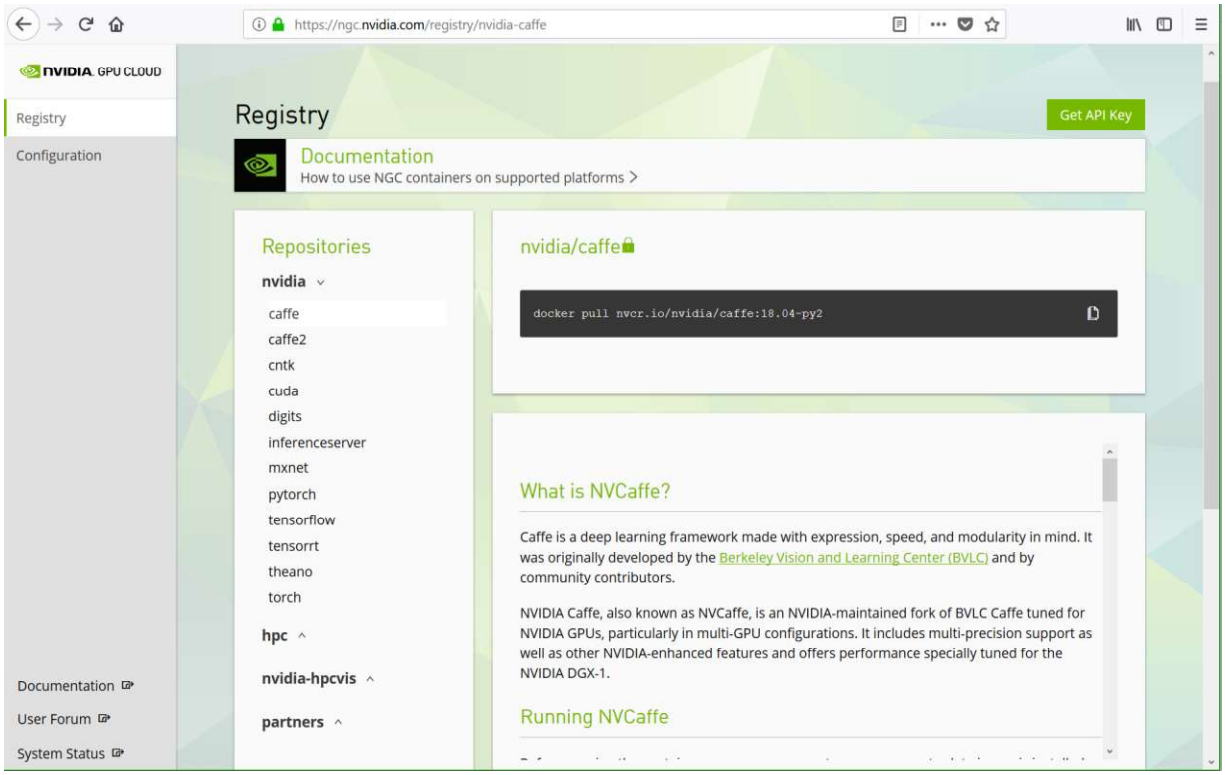


Figure 21: NVIDIA GPU Cloud registry page

In Figure 21, clicking the button "`Get API Key`" on the top right will take the user to the configuration page as shown in Figure 22. "Generate API Key" will generate an API key that is specific to the registered user.

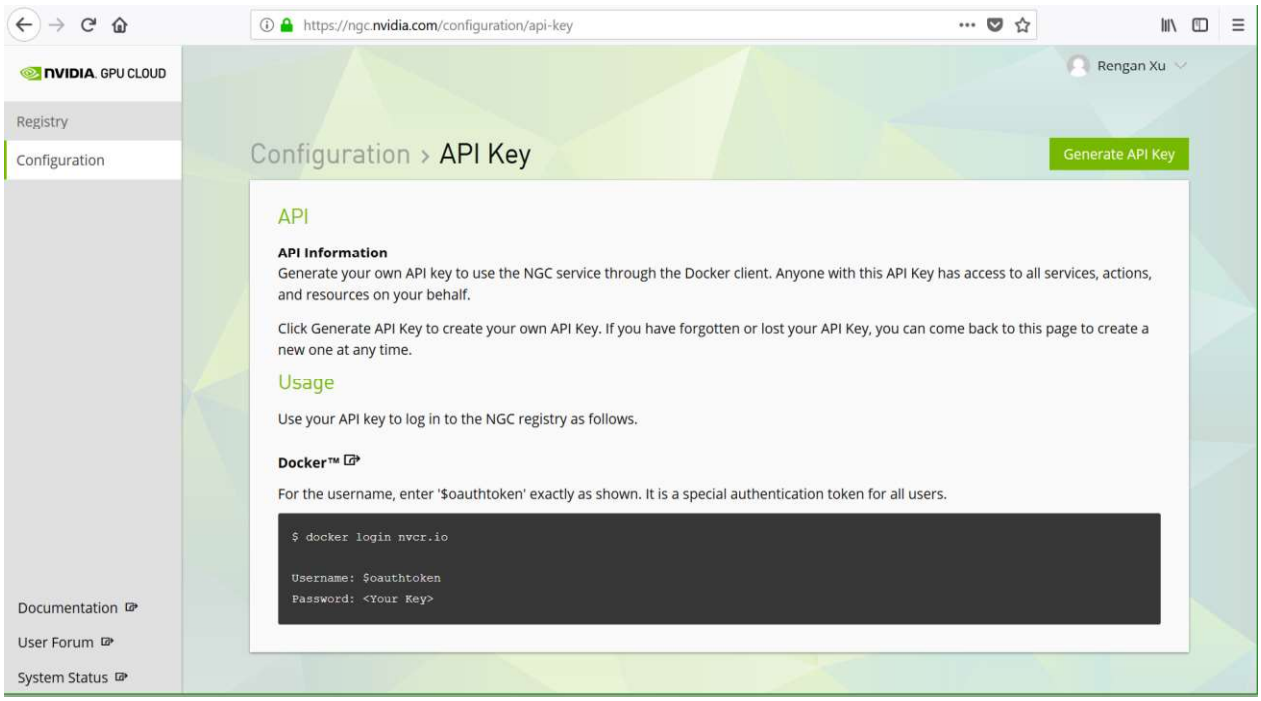


Figure 22: NVIDIA GPU Cloud configuration page

A user can download the Docker image of a framework directly with the command in Figure 23.

```
 $ docker login nvcr.io

  Username: $oauthtoken

  Password:

  Login Succeed

 $ docker pull nvcr.io/nvidia/mxnet:17.10

 $ docker images

  REPOSITORY                    TAG                 IMAGE ID            CREATED
SIZE

  nvcr.io/nvidia/mxnet          17.10               9a4558c1fa1a        7 months ago
2.6GB

 $ nvidia-docker run --interactive --detach 9a4558c1fa1a

 $ docker exec 5d2405dbb31c python /opt/mxnet/example/image-
classification/train_mnist.py --gpu 0 --num-epochs 10
```

Figure 23: Steps to download a Docker image from NGC

If the user wants to use Singularity container instead of Docker containers, Figure 24 shows how to create a Singularity image from the Docker image downloaded from NGC. The user must provide a set of environment variables which includes the user's NGC API key, and then use `sregistry` to convert a Docker image to Singularity image. After the Singularity image is generated, the user can use "`exec`" or other commands to execute the script `run_script.sh` within the created Singularity container. For more Singularity commands, please refer to Singularity User Guide.

```
  $ export SREGISTRY_NVIDIA_BASE=``ngcr.io”

  $ export SREGISTRY_CLIENT=nvidia

  $ export SREGISTRY_NVIDIA_USERNAME='$oauthtoken’

  $ export SREGISTRY_NVIDIA_TOKEN='[NGC_API_KEY]’

  $ sregistry pull nvidia://tensorflow:17.11

   Success! /home/user1/.singularity/shub/nvidia-tensorflow:17.11.simg

  $ singularity exec /home/user1/.singularity/shub/nvidia-tensorflow:17.11.simg
run_script.sh
```

Figure 24: The commands of creating Singularity image from NGC

5 The Data Scientist Portal

The data scientist portal was developed by Dell EMC and it simplifies the users' work allowing model implementation, training and inference tests to be run in Jupyter Notebook or directly from the Linux terminal. The Jupyter Notebook allows a user to write explanatory text and intersperse it with raw codes and the tables and figures that those codes generate. It allows a user to directly execute the codes that are embedded in the document that is being created or has been created. The portal also allows multiple users to share the same resources in the cluster. In the current version, the resources allocated by the portal is limited to only one node. If the user needs to use resources that span across two or more nodes, the Slurm job scheduler as described in Section 5.3 can be used instead. A deeper dive into the portal and its components is listed in Section 2.7.

5.1 Creating and Running a Notebook

A user can access the portal via a web browser on port 8000 (Figure 25). The user can specify the address either through "`localhost`" if logging to the cluster head node, or through the cluster head node IP address without logging into the cluster. The username and password are the same as those for logging to the cluster.

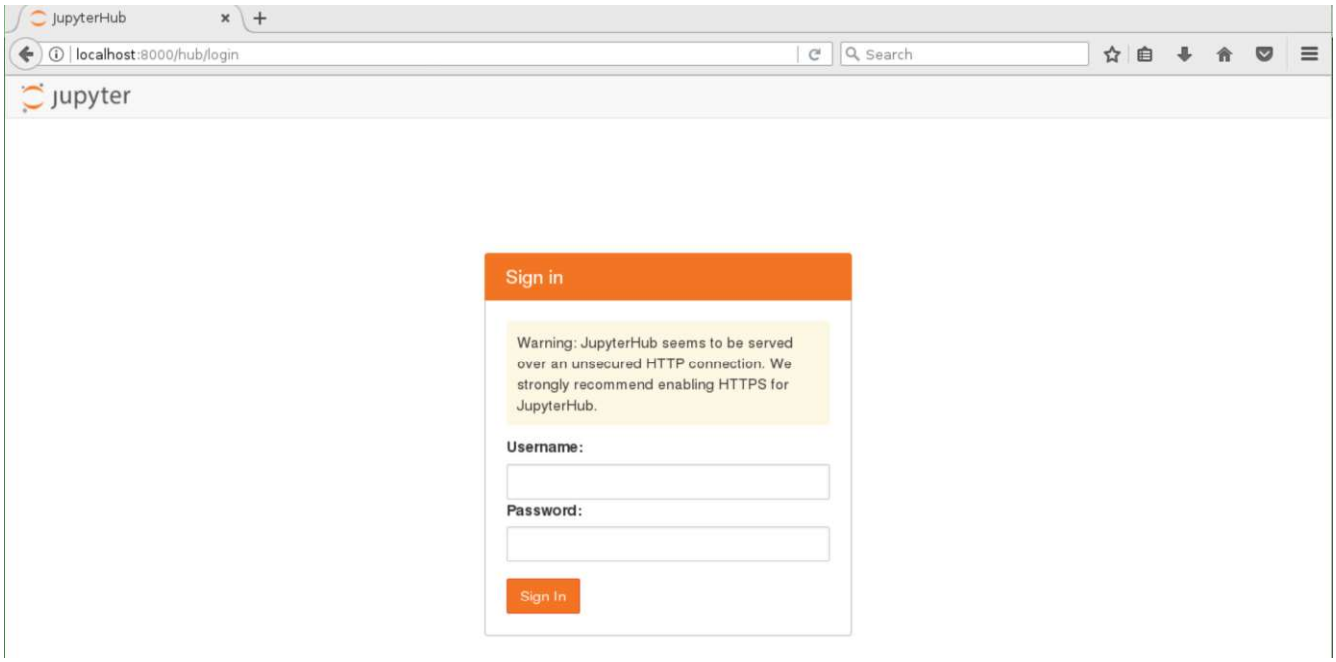


Figure 25: Portal login screen

After logging in, the page is forwarded to the server page (Figure 26). It lists the available server.

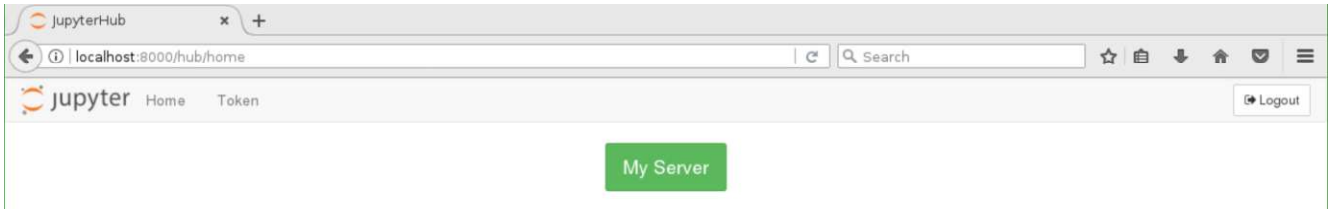


Figure 26: Portal server page

After clicking the "`My Server`" button, the user is forwarded to a list of instance options (Figure 27). The user is given options to choose how many GPUs to be shared. For each allocated GPU, 8 CPU cores and 48GB system memory are allocated. And by default the session time is allocated to 8 hours.

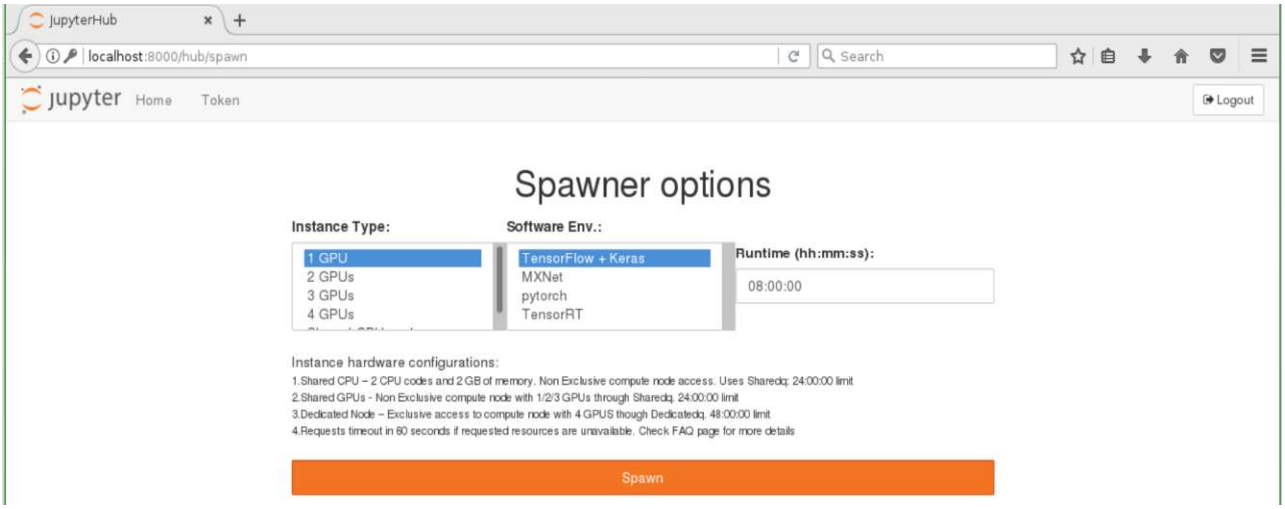


Figure 27: Portal instance screen

After the resources are allocated, the portal goes to the landing screen (Figure 28). It includes the folder "`templates`" which contains a set of template files of running different Deep Learning frameworks and libraries, and "`isilon`" which is the mount point where Isilon storage is mounted.

Figure 28: Portal landing screen

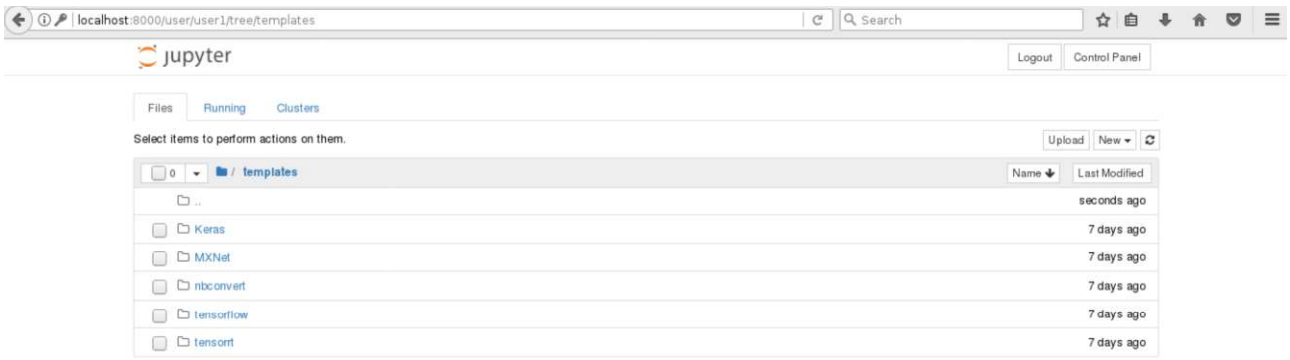


Figure 29: The list of available framework Jupyter Notebook templates

Clicking the "`New`" button in Figure 28 displays a list of kernels (Figure 30). A user can choose Python 2 or Python 3 kernel depending on the user's preferred Python version. The portal also provides R kernel so that the user can use R package. In "`Other`" kernels, the user can create a text file, a folder, launch a terminal or

launch a Tensorboard. Figure 31 is a screenshot after launching a terminal. How to use Tensorboard is shown in Section 5.2.

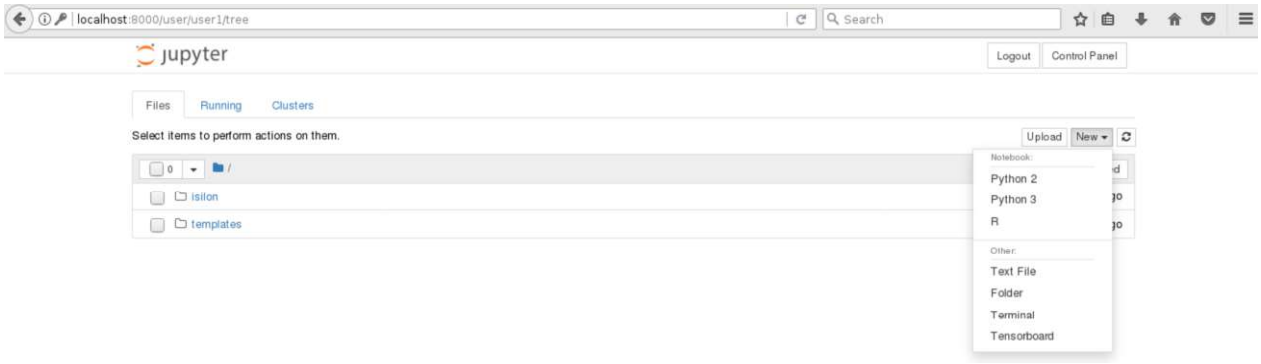


Figure 30: Portal kernel list

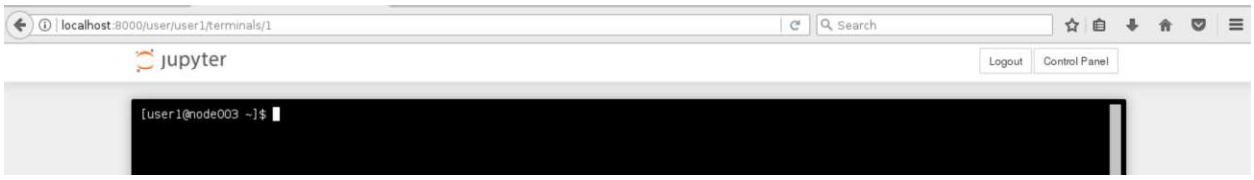


Figure 31: Portal terminal kernel

In the list of framework templates, choosing "`tensorflow`" will bring up the MNIST handwritten digits classification example shown in Figure 32. Now click on "`Kernel`" tab and select "`Restart and Run all`" as shown in Figure 33, the training of the handwritten digits classification will start. An example output is shown in Figure 34.

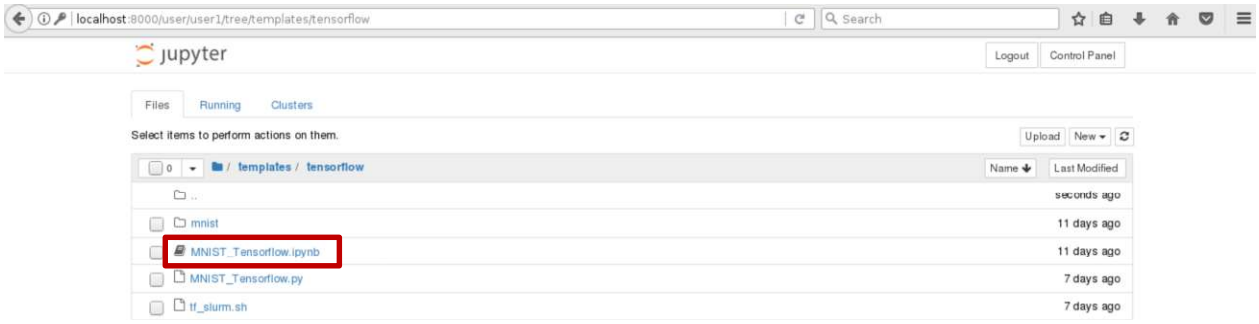


Figure 32: TensorFlow notebook

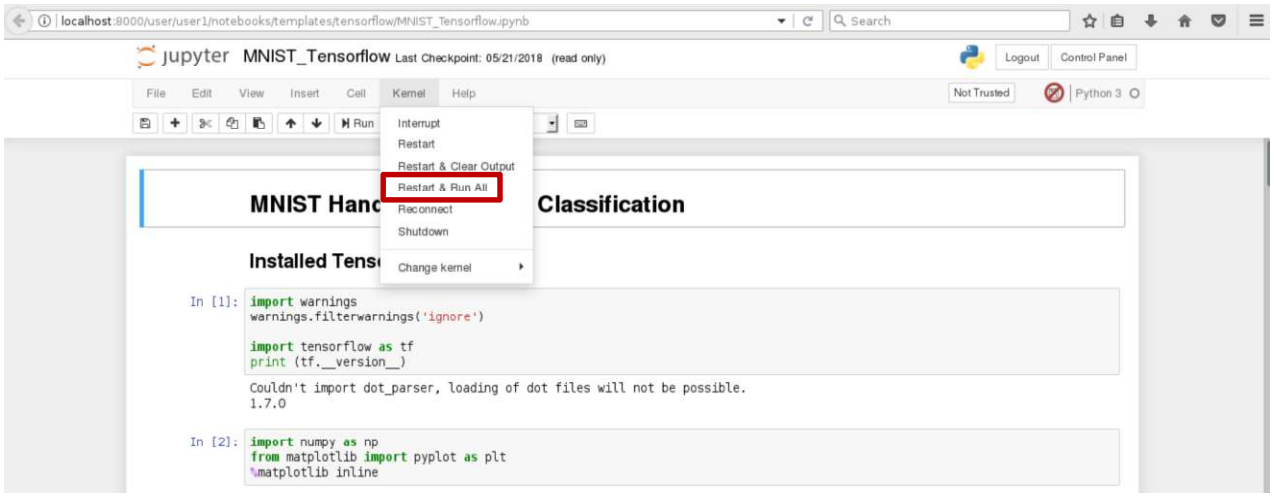


Figure 33: TensorFlow notebook handwritten digits classification

```
Epoch 1 Global Step 1, Training Accuracy 0.1953
Test Accuracy 0.2130
Epoch 1 Global Step 101, Training Accuracy 0.9688
Test Accuracy 0.9571
Epoch 1 Global Step 201, Training Accuracy 0.9297
Test Accuracy 0.9725
Epoch 1 Global Step 301, Training Accuracy 0.9766
Test Accuracy 0.9741
Epoch 1 Global Step 401, Training Accuracy 1.0000
Test Accuracy 0.9782
Epoch 2 Global Step 430, Training Accuracy 1.0000
Test Accuracy 0.9813
Epoch 2 Global Step 530, Training Accuracy 0.9844
Test Accuracy 0.9824
Epoch 2 Global Step 630, Training Accuracy 0.9922
Test Accuracy 0.9832
Epoch 2 Global Step 730, Training Accuracy 0.9922
Test Accuracy 0.9853
Epoch 2 Global Step 830, Training Accuracy 0.9766
Test Accuracy 0.9869
Epoch 3 Global Step 859, Training Accuracy 0.9766
Test Accuracy 0.9860
Epoch 3 Global Step 959, Training Accuracy 1.0000
Test Accuracy 0.9874
Epoch 3 Global Step 1059, Training Accuracy 0.9766
Test Accuracy 0.9874
Epoch 3 Global Step 1159, Training Accuracy 0.9922
Test Accuracy 0.9889
Epoch 3 Global Step 1259, Training Accuracy 1.0000
Test Accuracy 0.9885
```

Figure 34: TensorFlow notebook handwritten digits classification output

After a user starts a kernel, the kernel will keep running until being stopped. To stop a kernel, the user needs to click the "`Control Panel`" in Figure 30, then the page will go to the page in Figure 35. After clicking "`Stop My Server`", the kernel will be stopped.

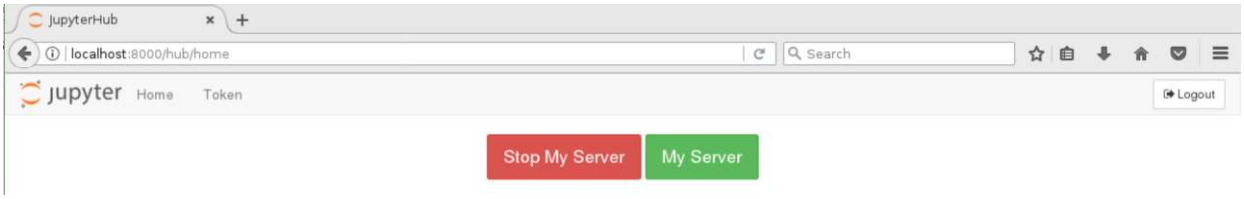


Figure 35: Stop the server

5.2 Tensorboard Integration

Besides the TensorFlow framework, the portal also provides the Tensorboard visualization tool. The Tensorboard is used to visualize the TensorFlow computation graph, plot quantitative metrics about the execution of a graph, and show some other additional data. To use Tensorboard, the user needs to use TensorFlow FileWriter API to serialize the wanted data into a directory. Figure 36 shows an example in which the TensorFlow outputs the serialized data into the folder "`logs`".

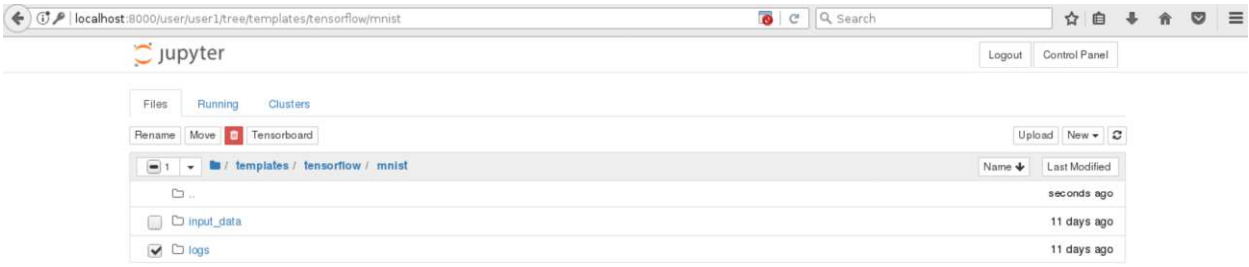


Figure 36: An example of using Tensorboard

After all serialized data are written into the folder "`logs`", the user can choose that folder and then Tensorboard button will be shown up. Then the user clicks "`Tensorboard`", Tensorboard will output all information in a separate tab as shown in Figure 37. The output result in Figure 37 is only an example. A user may get a different result depending on what information are written to the logs.

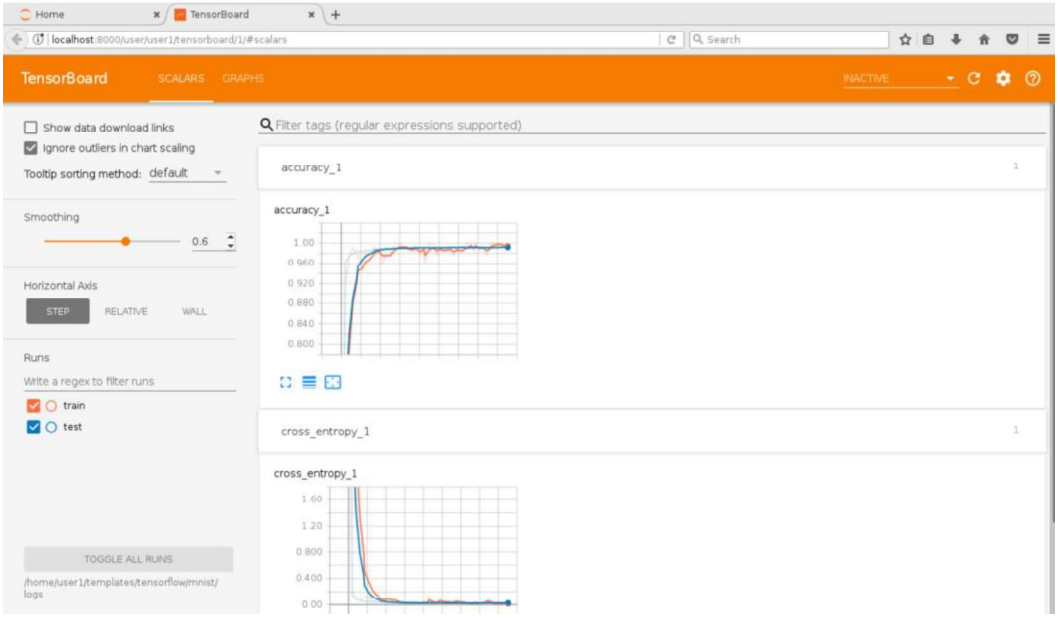


Figure 37: An example Tensorboard output

5.3 Slurm Scheduler

Section 5.1 describes the data scientist portal, but the current portal version can allocate resources only within one node. To use resources on multiple nodes, the user can use Slurm job scheduler. The Slurm scheduler is used to manage the resource allocation and job submissions for all users in a cluster. To use Slurm, the user needs to submit a script on the cluster head node that specifies what resources are required and what job should be executed on those resources once allocated.

Figure 38 shows an example Slurm script. In this example, the user asks for 2 nodes (`-N`) and 4 tasks per node (`--ntasks-per-node`), resulting in 8 tasks in total (`-n`). One task implies one CPU process. Nodes that include four GPUs are requested using the `-gres=gpu:4` descriptor. This job will be submitted to the "`sxm2`" queue (`-p`).

The job itself will run two commands: "`hostname`" and "`p2pBandwidthLatencyTest`" from the CUDA SDK. Since running this job requires CUDA toolkit, the user also needs to load the module files for "`shared`" and "`cuda91/toolkit`" that set the path and environment variables needed to execute the `p2pBandwidthLatencyTest` test.

The command "`sinfo`" can be used to check on the types of resources available on the system. Figure 39 shows an example output before running sample.job. After checking enough resources are available, the script can be run with the command "`sbatch sample.job`". The command "`squeue`" can be used to query the status of running jobs. Figure 40 shows an example output after running "`squeue`" to check the status of the current jobs. Note the output script name includes the Slurm job number that was visible in the squeue command output. An example output file name for Figure 38 is slurm-27325.out where 27325 is an example job id. Figure 41 shows an example content of this file. The output file displays the `hostname` and the output of `p2pBandwidthLatencyTest`. Note that although the script in Figure 38 allocated 8 processes, it only used one process in the execution command. To use all processes, the user can use MPI or other multi-process programming model.

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -n 8
#SBATCH --ntasks-per-node 4
#SBATCH -p sxm2
#SBATCH --gres=gpu:4
module load shared
module load cuda91/toolkit
hostname
/cm/shared/apps/cuda91/sdk/9.1.85/bin/x86_64/linux/release/p2pBandwidthLatencyTest
```

Figure 38: An example Slurm job script named sample.job

```
PARTITION  AVAIL  TIMELIMIT NODES  STATE  NODELIST
hipri*      up   12:00:00   17     down*  node[001-007,009-010,012,015-016,018-019]
sxm2        up   12:00:00   2      idle   node[008,014]
iq          up   6:00:00    2      down*  node[011,013]
requestq    up   infinite   2      idle   node[008,014]
```

Figure 39: An example output of the command sinfo before running sample.job

```
JOBID     PARTITION     NAME     USER     ST      TIME  NODES NODELIST(REASON)
 27325     sxm2        sample.j  root      R      0:01    2    node[008,014]
```

Figure 40: An example output of the command squeue showing the job status

```
node008
[P2P (Peer-to-Peer) GPU Bandwidth Latency Test]
Device: 0, Tesla V100-SXM2-16GB, pciBusID: 1a, pciDeviceID: 0, pciDomainID:0
Device: 1, Tesla V100-SXM2-16GB, pciBusID: 1c, pciDeviceID: 0, pciDomainID:0
Device: 2, Tesla V100-SXM2-16GB, pciBusID: 1d, pciDeviceID: 0, pciDomainID:0
Device: 3, Tesla V100-SXM2-16GB, pciBusID: 1e, pciDeviceID: 0, pciDomainID:0
Device=0 CAN Access Peer Device=1
Device=0 CAN Access Peer Device=2
Device=0 CAN Access Peer Device=3
Device=1 CAN Access Peer Device=0
Device=1 CAN Access Peer Device=2
Device=1 CAN Access Peer Device=3
Device=2 CAN Access Peer Device=0
Device=2 CAN Access Peer Device=1
Device=2 CAN Access Peer Device=3
Device=3 CAN Access Peer Device=0
Device=3 CAN Access Peer Device=1
Device=3 CAN Access Peer Device=2

***NOTE: In case a device doesn't have P2P access to other one, it falls back to normal
memcopy procedure.
So you can see lesser Bandwidth (GB/s) in those cases.

P2P Connectivity Matrix
     D\D     0     1     2     3
     0       1     1     1     1
     1       1     1     1     1
     2       1     1     1     1
     3       1     1     1     1
Unidirectional P2P=Disabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 741.22   9.94   9.91   9.94
     1   9.86 739.82   9.96   9.97
     2   9.83   9.94 737.03   9.98
     3   9.84   9.91   9.85 739.82
Unidirectional P2P=Enabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 741.22  47.72  47.73  47.86
     1  47.86 738.42  34.18  38.74
     2  39.57  47.71 738.42  47.82
     3  47.82  47.72  47.72 739.82
Bidirectional P2P=Disabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 751.92  10.42  10.41  10.50
     1  10.48 754.83  10.38  10.42
     2  10.43  10.43 759.97  10.35
     3  10.42  10.43  10.60 759.97
```

```
Bidirectional P2P=Enabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 754.83  75.25  78.63  78.80
     1  78.45 755.56  95.58  95.60
     2  95.48  95.48 755.56  78.47
     3  78.23  95.79  95.81 754.83
P2P=Disabled Latency Matrix (us)
   D\D     0      1      2      3
     0   3.61  18.35  18.32  18.31
     1  18.34   3.59  18.32  18.32
     2  18.35  18.34   3.57  18.31
     3  18.32  18.33  18.30   3.55
P2P=Enabled Latency Matrix (us)
   D\D     0      1      2      3
     0   3.58   6.68   6.65   6.75
     1   6.73   3.64   6.63   6.73
     2   6.42   6.45   3.56   6.42
     3   6.44   6.47   6.48   3.58

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary
when GPU Boost is enabled.
```

Figure 41: An example output for the after running sample.job. The file name is slurm-27325.out.

6 Conclusions and Future Work

This document describes the first integrated Dell EMC Ready Solutions for AI - Deep Learning with NVIDIA. The goal of this solution is to provide a complete, tuned and supported solution for Deep Learning training and inference use cases. The solution takes into account the ideal compute, storage, network, and software configuration for this workload. It includes tools that improve the usability of the system for data scientists, like the Data Scientist Portal that makes it easy for users to develop, train and run inferencing for Deep Learning models. The solution pre-installs a set of frameworks and libraries that are necessary for Deep Learning in addition to all other the software (operating system, drivers, libraries, management utilities, resource manager) required on a cluster.

This white paper provides comprehensive benchmark performance results and analysis that demonstrate the potential and usefulness of the proposed solution.

In the future work, the new 32GB V100 GPUs will be evaluated. Also more types of datasets and neural network models will be benchmarked and analyzed.