

Dell EMC Isilon, PowerSwitch, and NVIDIA DGX-2 Systems for Deep Learning

This document demonstrates how the Dell EMC Isilon F800 all-flash scale-out NAS, Dell EMC™ PowerSwitch™ S5232F-ON 100 Gbps switches and NVIDIA® DGX-2™ systems with NVIDIA Tesla® V100 GPUs can be used to accelerate and scale deep learning training workloads. The results of industry-standard image classification benchmarks using TensorFlow are included.

February 2020

The information in this publication is provided "as is." Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

This document may contain certain words that are not consistent with Dell's current language guidelines. Dell plans to update the document over subsequent future releases to revise these words accordingly.

This document may contain language from third party content that is not under Dell's control and is not consistent with Dell's current guidelines for Dell's own content. When such third party content is updated by the relevant third parties, this document will be revised accordingly.

Copyright © 2021 Dell Inc. or its subsidiaries. All Rights Reserved. Dell Technologies, Dell, EMC, Dell EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners. [3/9/2021] [Technical White Paper] [H18079]

Table of Contents

Revisions.....	5
Executive summary.....	5
Audience	5
Introduction.....	5
Deep learning dataflow	5
Solution architecture	7
OVERVIEW	7
STORAGE: DELL EMC ISILON F800.....	7
Storage tiering.....	9
OneFS caching	9
Locks and concurrency.....	9
NETWORKING: DELL EMC POWERSWITCH S5232F-ON SWITCH	10
COMPUTE: NVIDIA DGX-2 SYSTEM.....	10
BILL OF MATERIALS.....	10
SOFTWARE VERSIONS.....	11
Deep learning training performance and analysis.....	11
BENCHMARK METHODOLOGY	11
BENCHMARK RESULTS.....	13
SYSTEM METRICS.....	14
MEASUREMENT OF NETWORK I/O BETWEEN DGX-2 SYSTEMS.....	16
UNDERSTANDING FILE CACHING.....	18
UNDERSTANDING THE TRAINING PIPELINE	18
NVIDIA COLLECTIVE COMMUNICATION LIBRARY (NCCL)	19
Storage-only performance.....	20
STORAGE NETWORK PERFORMANCE USING IPERF	20
STORAGE-ONLY PERFORMANCE USING FIO	20
STORAGE-ONLY PERFORMANCE USING TENSORFLOW.....	20
Solution sizing guidance	23
Conclusions.....	24
Acknowledgements	25
Appendix – System configuration	26
ISILON	26
Configuration.....	26
Configuring automatic storage tiering	26
Testing automatic storage tiering.....	28
DELL EMC POWERSWITCH S5232F-ON DATA SWITCHES.....	29
NVIDIA DGX-2 SYSTEM.....	29
INSTALL AI BENCHMARK UTILITIES.....	31
ISILON VOLUME MOUNTING	32
Appendix – Benchmark setup	32

CREATING THE IMAGENET TFRECORD DATASETS.....	32
OBTAIN THE TENSORFLOW BENCHMARKS	32
START TENSORFLOW CONTAINERS.....	33
Appendix – Monitoring Isilon performance.....	34
INSIGHTIQ	34
ISILON STATISTICS CLI	34
Appendix – Isilon performance testing with iPerf and FIO.....	35
IPERF	35
Using iPerf to test Isilon to DGX-2 system performance	35
Using iPerf to test DGX-2 system storage network performance	36
FIO.....	36
Appendix – Isilon performance testing with TensorFlow	37
Appendix – Switch configuration	37
Appendix – Collecting system metrics with Prometheus and Grafana	42
References.....	44

Revisions

Date	Description	Author
February 2020	Initial release	Claudio Fahey

Executive summary

Deep learning (DL) techniques have enabled great successes 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 larger datasets and increased computational power. To be effective at enterprise scale, the computational intensity of DL requires highly powerful and efficient parallel architectures. The choice and design of the system components, carefully selected and tuned for DL use-cases, can have a big impact on the speed, accuracy and business value of implementing artificial intelligence (AI) techniques.

In such a complex environment, it is critical that organizations be able to rely on vendors that they trust. Over the last few years, Dell Technologies and NVIDIA have established a strong partnership to help organizations fast-track their AI initiatives. Our partnership is built on the philosophy of offering flexibility and informed choice across a broad portfolio which combines best of breed GPU accelerated compute, scale-out storage, and networking.

This paper focuses on how Dell EMC Isilon F800 all-flash scale-out NAS and Dell EMC PowerSwitch S5232F-ON switches accelerate AI innovation by delivering the performance, scalability and concurrency to complement the requirements of NVIDIA DGX-2™ systems for high performance AI workloads.

Audience

This document is intended for organizations interested in simplifying and accelerating DL solutions with advanced computing and scale-out data management solutions. Solution architects, system administrators and other interested readers within those organizations constitute the target audience.

Introduction

DL is an area of AI which uses artificial neural networks to enable accurate pattern recognition of complex real-world patterns by computers. These new levels of innovation have applicability across nearly every industry vertical. Some of the early adopters include advanced research, precision medicine, high tech manufacturing, advanced driver assistance systems (ADAS) and autonomous driving. Building on these initial successes, AI initiatives are springing up in various business units, such as manufacturing, customer support, life sciences, marketing, and sales. Gartner [predicts](#) that AI augmentation will generate \$2.9 trillion in business value by 2021 alone. Organizations are faced with a multitude of complex choices related to data, analytic skill-sets, software stacks, analytic toolkits, and infrastructure components; each with significant implications on the time to market and the value associated with these initiatives.

In such a complex environment, it is critical that organizations be able to rely on vendors that they trust. Over the last few years, Dell Technologies and NVIDIA have established a strong partnership to help organizations accelerate their AI initiatives. Our partnership is built on the philosophy of offering flexibility and informed choice across an extensive portfolio. Together our technologies provide the foundation for successful AI solutions which drive the development of advanced DL software frameworks, deliver massively parallel compute in the form of NVIDIA GPUs for parallel model training and scale-out file systems to support the concurrency, performance, and capacity requirements of unstructured image and video data sets.

This document focuses on the latest step in the Dell Technologies and NVIDIA collaboration, a new AI reference architecture with Isilon F800 storage, PowerSwitch S5232F-ON switches, and DGX-2 systems for DL workloads. This new offer gives customers more flexibility in how they deploy scalable, high performance DL. The results of multiple industry standard image classification benchmarks using TensorFlow are included.

Deep learning dataflow

As visualized in Figure 1, DL usually consist of two distinct workflows, model development and inference.

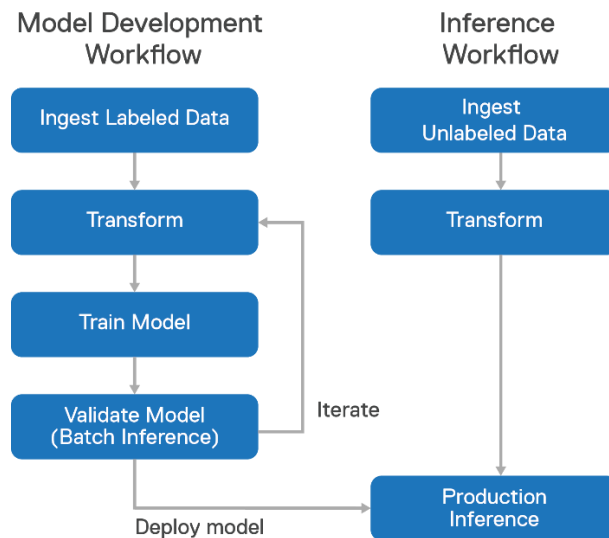


Figure 1: Common DL Workflows: Model development and inference

Note: The Isilon storage and DGX-2 system architecture is optimized for the model development workflow which consists of the model training and the batch inference validation steps. It is not intended for and nor was it benchmarked for production inference.

The workflow steps are defined and detailed below.

1. **Ingest Labeled Data** — The labeled data (e.g. images and their labels which indicate whether the image contains a dog, cat, or horse) are ingested into the DL system.
2. **Transform** — Transformation includes all operations that are applied to the labeled data before they are passed to the DL algorithm. It is sometimes referred to as preprocessing. For images, this often includes file parsing, JPEG decoding, cropping, resizing, rotation, and color adjustments. Transformations can be performed on the entire dataset ahead of time, storing the transformed data on disk. Many transformations can also be applied in a training pipeline, avoiding the need to store the intermediate data.
3. **Train Model** — The model parameters (edge weights) are learned from the labeled data using the stochastic gradient descent optimization method. In the case of image classification, there are several prebuilt structures of neural networks that have been shown to work well. To provide an example, Figure 2 shows the high-level structure of the Inception-v3 model which contains nearly 25 million parameters that must be learned. In this diagram, images enter from the left and the probability of each class comes out on the right.

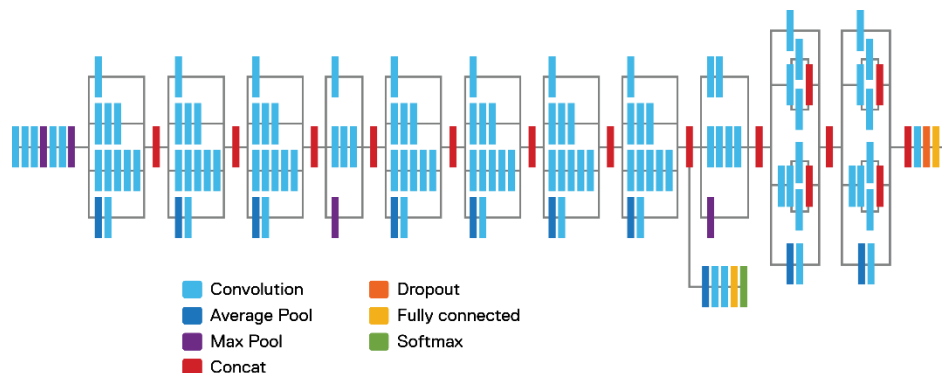


Figure 2: Inception-v3 model architecture.

6. **Validate Model** — Once the model training phase completes with a satisfactory accuracy, you'll want to measure the accuracy of it on validation data – data that the model training process has not seen. This is done by using the

trained model to make inferences from the validation data and comparing the result with the correct label. This is often referred to as inference but keep in mind that this is a distinct step from production inference.

7. **Production Inference** — The trained and validated model is then often deployed to a system that can perform real-time inference. It will accept as input a single image and output the predicted class (dog, cat, horse). In some cases, inputs are batched for higher throughput but higher latency.

Solution architecture

OVERVIEW

Figure 3 illustrates the reference architecture showing the key components that made up the solution as it was tested and benchmarked. Note that in a customer deployment, the number of DGX-2 systems and Isilon storage nodes will vary and can be scaled independently to meet the requirements of the specific DL workloads. Refer to the Solution sizing guidance section for details.

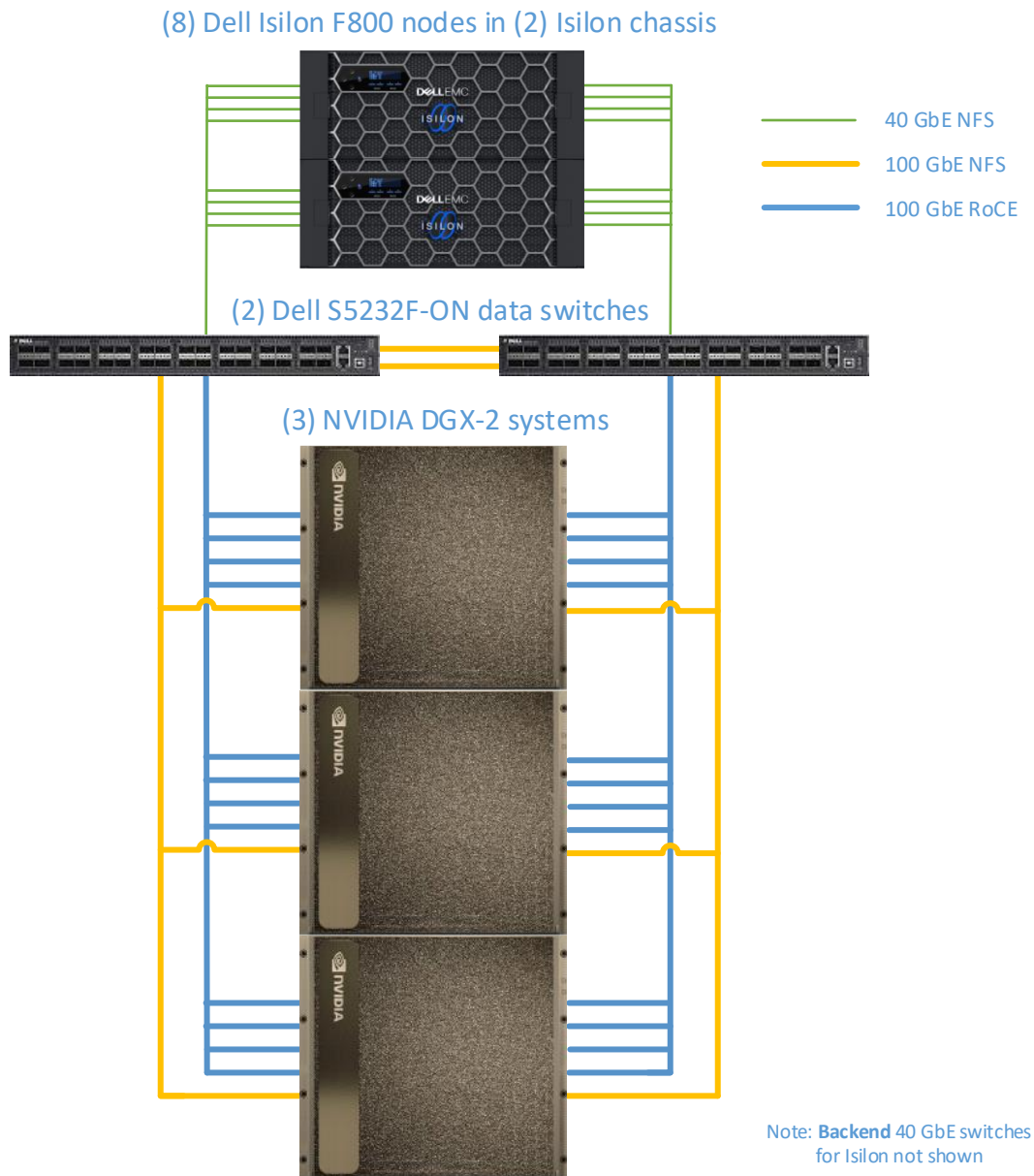


Figure 3: Reference Architecture

STORAGE: DELL EMC ISILON F800

Dell EMC Isilon F800 represents the sixth generation of hardware built to run the well-proven and massively scalable OneFS operating system. Each F800 chassis, shown in Figure 4, contains four storage nodes, 60 high-performance

solid-state drives (SSDs) and eight 40 GbE network connections. OneFS combines up to 252 nodes in 63 chassis into a single high-performance file system designed to handle the most intense I/O workloads such as DL. As performance and capacity demands increase, both can be scaled-out simply and non-disruptively, allowing applications and users to continue working.



Figure 4: Isilon F800 chassis, containing four storage nodes

In the solution tested in this document, eight F800 nodes, in two chassis, were used.

Dell EMC Isilon F800 has the following features.

Low latency, high throughput, and massively parallel I/O for AI:

- Up to 250,000 file IOPS per chassis, up to 15.75 million IOPS per cluster
- Up to 15 GB/s throughput per chassis, up to 945 GB/s per cluster
- 96 TB to 924 TB raw flash capacity per chassis; up to 58 PB per cluster (all-flash)

This shortens time for training and testing analytical models for data sets from tens of TBs to tens of PBs on AI platforms such as RAPIDS, TensorFlow, SparkML, Caffe, or proprietary AI platforms.

The ability to run AI in-place on data using multi-protocol access:

- Multi-protocol support such as SMB, NFS, HTTP, and native HDFS to maximize operational flexibility

This eliminates the need to migrate/copy data and results over to a separate AI stack. Organizations can perform DL and run other IT apps on the same data already on Isilon by adding additional Isilon nodes to an existing cluster.

Enterprise grade features out-of-box:

- Enterprise data protection and resiliency
- Robust security options

This enables organizations to manage AI data lifecycle with minimal cost and risk, while protecting data and meeting regulatory requirements.

Extreme scale:

- Seamlessly tier between All Flash, Hybrid, and Archive nodes via SmartPools
- Grow-as-you-go scalability with up to 58 PB flash capacity per cluster
- New nodes can be added to a cluster simply by connecting power, back-end Ethernet and front-end Ethernet
- As new nodes are added, storage capacity, throughput, IOPS, cache, and CPU grow
- Up to 63 chassis (252 nodes) may be connected to form a single cluster with a single namespace and a single coherent cache
- Up to 85% storage efficiency to reduce costs
- Optional data de-dup and compression enabling up to a 3:1 data reduction

Organizations can achieve AI at scale in a cost-effective manner, enabling them to handle multi-petabyte datasets with high resolution content without re-architecture and/or performance degradation.

There are several key features of Isilon OneFS that make it an excellent storage system for DL workloads that require performance, concurrency, and scale. These features are detailed below.

Storage tiering

Dell EMC Isilon SmartPools software enables multiple levels of performance, protection, and storage density to co-exist within the same file system and unlocks the ability to aggregate and consolidate a wide range of applications within a single extensible, ubiquitous storage resource pool. This helps provide granular performance optimization, workflow isolation, higher utilization, and independent scalability – all with a single point of management.

SmartPools allows you to define the value of the data within your workflows based on policies and automatically aligns data to the appropriate price/performance tier over time. Data movement is seamless and with file-level granularity and control via automated policies, manual control or API, you can tune performance and layout, storage tier alignment and protection settings – all with minimal impact to your end-users.

Storage tiering has a very convincing value proposition, namely separating data according to its business value and aligning it with the appropriate class of storage and levels of performance and protection. Information Lifecycle Management techniques have been around for several years, but have typically suffered from the following inefficiencies: complex to install and manage, involves changes to the file system, requires the use of stub files, etc.

Dell EMC Isilon SmartPools is a next generation approach to tiering that facilitates the management of heterogeneous clusters. The SmartPools capability is native to the Isilon OneFS scale-out file system, which allows for unprecedented flexibility, granularity, and ease of management. In order to achieve this, SmartPools leverages many of the components and attributes of OneFS, including data layout and mobility, protection, performance, scheduling and impact management.

A typical Isilon cluster will store multiple datasets with different performance, protection, and price requirements. Generally, files that have been recently created and accessed should be stored in a hot tier while files that have not been accessed recently should be stored in a cold tier. Because Isilon supports tiering based on a file's access time, this can be performed automatically. For storage administrators that want more control, complex rules can be defined to set the storage tier based on a file's path, size, or other attributes.

All files on Isilon are always immediately accessible (read and write) regardless of their storage tier and even while being moved between tiers. The file system path to a file is not changed by tiering. Storage tiering policies are applied, and files are moved by the Isilon SmartPools job, which runs daily at 22:00 by default.

For more details, see [Storage Tiering with Dell EMC Isilon SmartPools](#).

OneFS caching

The OneFS caching infrastructure design is predicated on aggregating the cache present on each node in a cluster into one globally accessible pool of memory. This allows all the memory cache in a node to be available to every node in the cluster. Remote memory is accessed over an internal interconnect and has lower latency than accessing hard disk drives and SSDs.

For files marked with an access pattern of concurrent or streaming, OneFS can take advantage of prefetching of data based on heuristics used by the Isilon SmartRead component. This greatly improves sequential-read performance across all protocols and means that reads come directly from RAM within milliseconds. For high-sequential cases, SmartRead can very aggressively prefetch ahead, allowing reads of individual files at very high data rates.

For more details, see [OneFS SmartFlash](#).

Locks and concurrency

OneFS has a fully distributed lock manager that coordinates locks on data across all nodes in a storage cluster. The lock manager is highly extensible and allows for multiple lock personalities to support both file system locks as well as cluster-coherent protocol-level locks such as SMB share mode locks or NFS advisory-mode locks. OneFS also has support for delegated locks such as CIFS oplocks and NFSv4 delegations. Every node in a cluster is a coordinator for locking resources and a coordinator is assigned to lockable resources based upon an advanced hashing algorithm.

Efficient locking is critical to support the efficient parallel I/O profile demanded by many iterative DL workloads enabling concurrent file read access up into the millions.

For more details, see the [OneFS Technical Overview](#).

NETWORKING: DELL EMC POWERSWITCH S5232F-ON SWITCH

The Dell EMC PowerSwitch S5200-ON 25/100 GbE fixed switches (Figure 5) comprise Dell EMC’s latest disaggregated hardware and software data center networking solutions, providing state-of-the-art, high-density 25/100 GbE ports and a broad range of functionality to meet the growing demands of today’s data center environment. These innovative, next-generation open networking switches offer optimum flexibility and cost-effectiveness for web 2.0, enterprise, mid- market and cloud service provider with demanding compute and storage traffic environments. This series supports RDMA over Converged Ethernet (RoCE) which allows a GPU to communicate with a NIC directly across the PCIe bus, without involving the CPU. Both RoCE v1 and v2 are supported.

The PowerSwitch S5232F-ON is a 1 RU switch with 32 QSFP28 ports that can provide 40 GbE and 100 GbE.



Figure 5: Dell EMC PowerSwitch S5232F-ON, 32 QSFP28 ports that support 40/100 GbE

COMPUTE: NVIDIA DGX-2 SYSTEM

The DGX-2 system (Figure 6) is a fully integrated, turnkey hardware and software system that is purpose-built for DL workflows. Each DGX-2 system is powered by 16 NVIDIA Tesla V100 GPUs that are interconnected using NVIDIA NVSwitch technology, which provides an ultra-high bandwidth low-latency fabric for inter-GPU communication. This topology is essential for multi-GPU training, eliminating the bottleneck that is associated with PCIe-based interconnects that cannot deliver linearity of performance as GPU count increases. The DGX-2 system is also equipped with eight high-bandwidth, low-latency network interconnects for multi-node clustering over RDMA-capable fabrics including RoCE and InfiniBand.



Figure 6: NVIDIA DGX-2 system with 16 Tesla V100 GPUs

NVIDIA GPU CLOUD

The NVIDIA GPU Cloud (NGC) container registry provides researchers, data scientists and developers with simple access to a comprehensive catalog of GPU-accelerated software for AI, DL, machine learning (ML) and HPC that take full advantage of NVIDIA DGX-2 systems. NGC provides containers for today’s most popular AI frameworks such as RAPIDS, Caffe2, TensorFlow, PyTorch, MXNet and TensorRT, which are optimized for NVIDIA GPUs. The containers integrate the framework or application, necessary drivers, libraries and communications primitives and they are optimized across the stack by NVIDIA for maximum GPU-accelerated performance. NGC containers incorporate the NVIDIA CUDA® Toolkit, which provides the NVIDIA CUDA Basic Linear Algebra Subroutines Library (cuBLAS), the NVIDIA CUDA Deep Neural Network Library (cuDNN), and much more. The NGC containers also include the NVIDIA Collective Communications Library (NCCL) for multi-GPU and multi-node collective communication primitives, enabling topology awareness for DL training. NCCL enables communication between GPUs inside a single DGX-2 system and across multiple DGX-2 systems.

BILL OF MATERIALS

Component	Purpose	Quantity
Dell EMC Isilon F800 96 TB SSD	Shared storage	2 4U chassis (8 nodes)

1 TB RAM		
Four 1 GbE, eight 40 GbE interfaces		
Celestica D4040 851-0259	Isilon back-end Ethernet switch	1
Dell EMC PowerSwitch S5232F-ON	Data switch	2
NVIDIA DGX-2 system	Compute server	3
16 Tesla V100-SXM3-32 GB GPUs		
Two 24-Core Intel Xeon Platinum 8168 2.70 GHz		
1.5 TB RAM		
One 2-port ConnectX5 NIC for storage		
100 GbE optical cable, QSFP28	DGX-2 system	30
100 GbE cable, QSFP28	data switch inter-switch links	2
Cable Mellanox 2m QSFP28 Copper 40 G	Isilon front-end	16
MC2206130-002		
Cable EMC AMPHENOL 3.0M 40G 038-002-066-02	Isilon back-end	8
1849, QSFP28		

Table 1: Bill of materials

SOFTWARE VERSIONS

Table 2 shows the software versions that were tested for this document.

Component	Version
AI Benchmark Util	https://github.com/claudiofahey/ai-benchmark-util/commit/99e8167
Dell EMC Isilon – OneFS	8.1.2.0 Patches: 8.1.2.0_KGA-RUP_2019-09_256178, 8.1.2.0_UGA-PATCH-INFRA_2019-09_255624, 8.1.2.0_UGA-RUP_2019-09_256176
Dell EMC Networking OS10 Enterprise	10.5.0.0.326
DGX-2 – Base OS	4.2.0
DGX-2 – BIOS	0.24
DGX-2 – Linux kernel	4.15.0-65-generic
DGX-2 – NVIDIA Driver	418.67
DGX-2 – Ubuntu	18.04.3 LTS
NVIDIA GPU Cloud TensorFlow Image	nvcr.io/nvidia/tensorflow:19.09-py3
TensorFlow	1.14.0
TensorFlow Benchmarks	https://github.com/claudiofahey/benchmarks/commit/31ea13f

Table 2: Software Versions

Deep learning training performance and analysis

BENCHMARK METHODOLOGY

In order to measure the performance of the solution, various benchmarks from the [TensorFlow Benchmarks](#) repository were executed. This suite of benchmarks performs training of an image classification convolutional neural network (CNN) on labeled images. Essentially, the system learns whether an image contains a cat, dog, car, train, etc. The well-known [ILSVRC2012](#) image dataset (often referred to as ImageNet) was used. This dataset contains 1,281,167 training images in 144.8 GB¹. All images are grouped into 1000 categories or classes. This dataset is commonly used by DL researchers for benchmarking and comparison studies.

The individual JPEG images in the ImageNet dataset were converted to 1024 TFRecord files (see Appendix – Benchmark setup). The TFRecord file format is a Protocol Buffers binary format that combines multiple JPEG image files together with their metadata (bounding box for cropping and label) into one binary file. It maintains the image compression offered by the JPEG format and the total size of the dataset remained roughly the same (148 GB). The average image size was 115 KB.

When running the benchmarks on the 148 GB dataset, it was found that the storage I/O throughput gradually decreased and became virtually zero after a few minutes. This indicated that the entire dataset was cached in the Linux buffer cache on each DGX-2 system. Of course, this is not surprising since each DGX-2 system has 1.5 TB of RAM and this workload did not significantly use RAM for other purposes. As real datasets are often significantly larger than this, we wanted to determine the performance with datasets that are not only larger than the DGX-2 system RAM, but larger than the 2 TB of

¹ All unit prefixes in this document use the SI standard (base 10) where 1 GB is 1 billion bytes.

coherent shared cache available across the eight-node Isilon cluster. To accomplish this, we simply made 150 exact copies of each TFRecord file, creating a 22.2 TB dataset.

In our own testing, a parallel Python MPI script was created to quickly create the copies utilizing all DGX-2 systems and Isilon nodes. But to illustrate the copy process, it was basically as simple as this:

```
cp train-00000-of-01024 train-00000-of-01024-copy-000
cp train-00000-of-01024 train-00000-of-01024-copy-001
cp train-00000-of-01024 train-00000-of-01024-copy-002
cp train-00001-of-01024 train-00001-of-01024-copy-000
cp train-00001-of-01024 train-00001-of-01024-copy-001
cp train-00001-of-01024 train-00001-of-01024-copy-002
...
cp train-01023-of-01024 train-01023-of-01024-copy-002
```

Having 150 copies of the exact same images doesn't improve training accuracy or speed but it does produce the same I/O pattern for the storage, network, and GPUs. Having identical files did not provide an unfair advantage as Isilon deduplication was not enabled and all images are reordered randomly (shuffled) in the input pipeline.

For this workload, it is straightforward to quantify the effect of caching. Essentially there are many threads reading the TFRecord files. Each thread picks a TFRecord file at random, reads it sequentially and completely and then it moves on to another TFRecord file at random. Sometimes, a thread will choose to read a file that another thread has recently read and that can be served from cache (either Isilon or Linux). The probability of this occurring and therefore the fraction of data served by cache, is simply the cache size divided by the dataset size. Using this calculation, we expect a 7% cache hit rate from the Linux cache on the DGX-2 system and a 9% cache hit rate from Isilon. Conversely, we can say that 91% of the dataset is read from Isilon's SSDs. Additionally, if the dataset were twice the size (44 TB), the Isilon SSD read rate would only increase by 4%.

One of the critical questions one has when trying to size a system is how fast the storage must be so that it is not a bottleneck. To answer this question, we take advantage of the fact that the Linux buffer cache can completely cache the entire 148 GB dataset. After performing several warm-up runs, the benchmark is executed, and it is confirmed that there is virtually zero NFS network I/O to the storage system. The image rate (images/sec) measured in this way accounts for the significant preprocessing pipeline as well as the GPU computation. To determine the throughput (bytes/sec) demanded by this workload, we simply multiply the images/sec by the average image size (115 KB). In the next section, results using this method are labeled Linux Cache.

To avoid confusion, the performance results using the synthetic benchmark mode are not reported in this document. In the synthetic mode of the benchmark, random images are generated directly in the GPU and training is based on these images. This mode is very useful to tune and understand parts of the training pipeline. However, it does not perform storage I/O, JPEG decoding, image resizing, or any other preprocessing.

There are a variety of ways to parallelize model training to take advantage of multiple GPUs across multiple servers. In our tests, we used Horovod. Horovod is a distributed DL training framework for TensorFlow and other DL frameworks. Horovod uses Message Passing Interface (MPI) for high-speed and low-latency communication between processes.

Prior to each execution of the benchmark, the L1 and L2 caches on Isilon were flushed with the command `isi_for_array isi_flush`. In addition, the Linux buffer cache was flushed on all DGX-2 systems by running `sync; echo 3 > /proc/sys/vm/drop_caches`. However, note that the training process will read the same files repeatedly and after just several minutes, much of the data will be served from one of these caches.

The command below was used to perform the ResNet-50 training with 48 GPUs.

```
mpirun \
--n 48 \
--allow-run-as-root \
--host dgx2-1:16,dgx2-2:16,dgx2-3:16 \
--report-bindings \
--bind-to none \
--map-by slot \
-x LD_LIBRARY_PATH \
-x PATH \
-mca plm_rsh_agent ssh \
-mca plm_rsh_args "-p 2222" \
-mca pml obl \
-mca btl ^openib \
-mca btl_tcp_if_include enp53s0\
```

```

-x NCCL_DEBUG=INFO \
-x NCCL_IB_HCA=mlx5 \
-x NCCL_IB_SL=4 \
-x NCCL_IB_GID_INDEX=3 \
-x NCCL_NET_GDR_READ=1 \
-x NCCL_SOCKET_IFNAME=^docker0,lo \
./round_robin_mpi.py \
python \
-u \
/mnt/isilon/data/tensorflow-benchmarks/scripts/tf_cnn_benchmarks/\
tf_cnn_benchmarks.py \
--model=resnet50 \
--batch_size=256 \
--batch_group_size=20 \
--num_batches=1000 \
--nodistortions \
--num_gpus=1 \
--device=gpu \
--force_gpu_compatible=True \
--data_format=NCHW \
--use_fp16=True \
--use_tf_layers=True \
--data_name=imagenet \
--use_datasets=True \
--num_intra_threads=1 \
--num_inter_threads=40 \
--datasets_prefetch_buffer_size=40 \
--datasets_num_private_threads=4 \
--train_dir=/mnt/isilon/data/train_dir/2019-10-24-14-53-59-resnet50 \
--sync_on_finish=True \
--summary_verbosity=1 \
--save_summaries_steps=100 \
--save_model_secs=600 \
--variable_update=horovod \
--horovod_device=gpu \
--data_dir=/mnt/isilon1/data/imagenet-scratch/tfrecords-150x \
--data_dir=/mnt/isilon2/data/imagenet-scratch/tfrecords-150x \
...
--data_dir=/mnt/isilon16/data/imagenet-scratch/tfrecords-150x

```

The script `round_robin_mpi.py` was used to select a single `--data_dir` parameter that distributed the processes across 16 different mount points. Note that when testing the Linux Cache performance, only a single mount point was used.

For different numbers of GPUs, only the `--np` parameter was changed. Note that the `-map-by` slot setting causes MPI to use all 16 GPUs (slots) on a DGX-2 system before it begins using the next DGX-2 system. For example, when testing 32 GPUs, only two DGX-2 systems are used.

For the other models, only the `--model` parameter was changed. However, for VGG-16, the batch size was set to 192.

The benchmark results in this section were obtained with eight Isilon F800 nodes in the cluster. Each result is the average of three executions.

BENCHMARK RESULTS

There are a few conclusions that we can make from the benchmarks represented in Figure 7.

- Image throughput and therefore storage throughput scale linearly from 16 to 48 GPUs.
- There is no significant difference in image throughput between Linux Cache and Isilon.

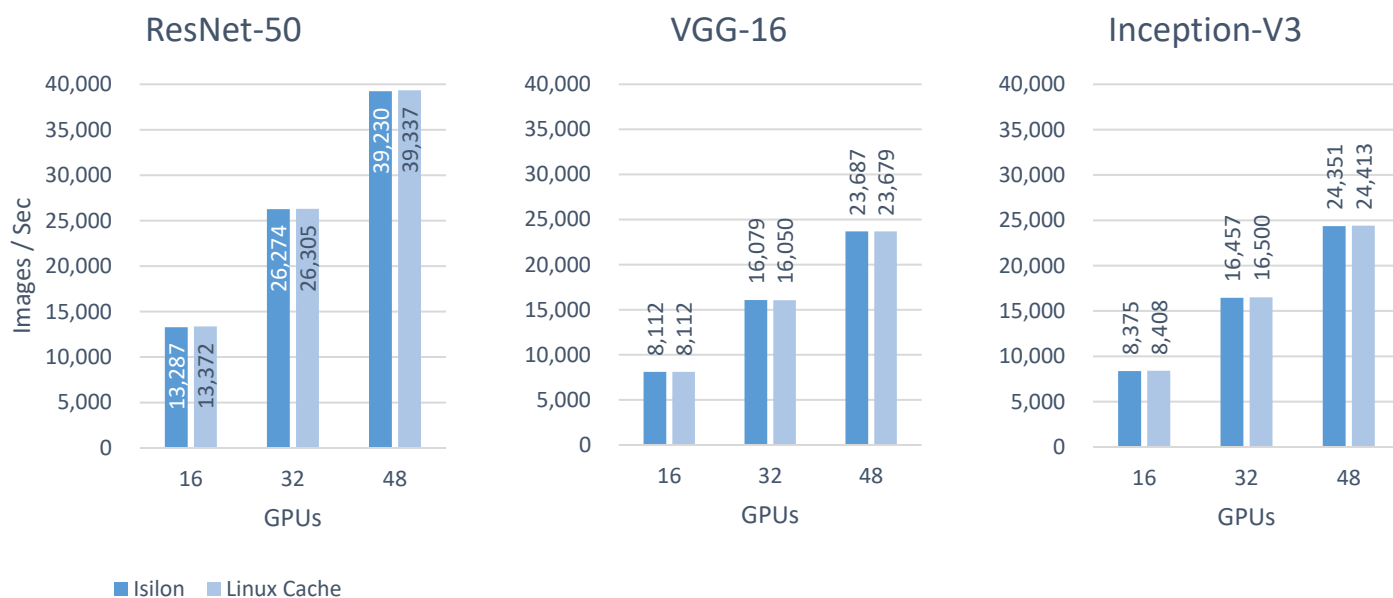


Figure 7: Model Development – Training Benchmark Results

SYSTEM METRICS

System metrics were collected and analyzed for all tests shown in Figure 7. In Figure 8, only metrics captured during three runs of ResNet-50 training on 48 GPUs are shown.

There are a few conclusions that we can make from the GPU and CPU metrics as represented in Figure 8.

- Each GPU had 97% utilization or higher. This indicates that the GPUs were fully utilized.
- The maximum CPU core utilization on the DGX-2 system was 70%. This occurred with ResNet-50.



Figure 8: GPU and CPU usage during ResNet-50 training on 48 GPUs

There are a few conclusions that we can make from the network metrics as represented in Figure 9.

- The maximum throughput that was pulled from Isilon occurred with ResNet-50 and 48 GPUs. The total storage throughput was 4501 MB/sec.

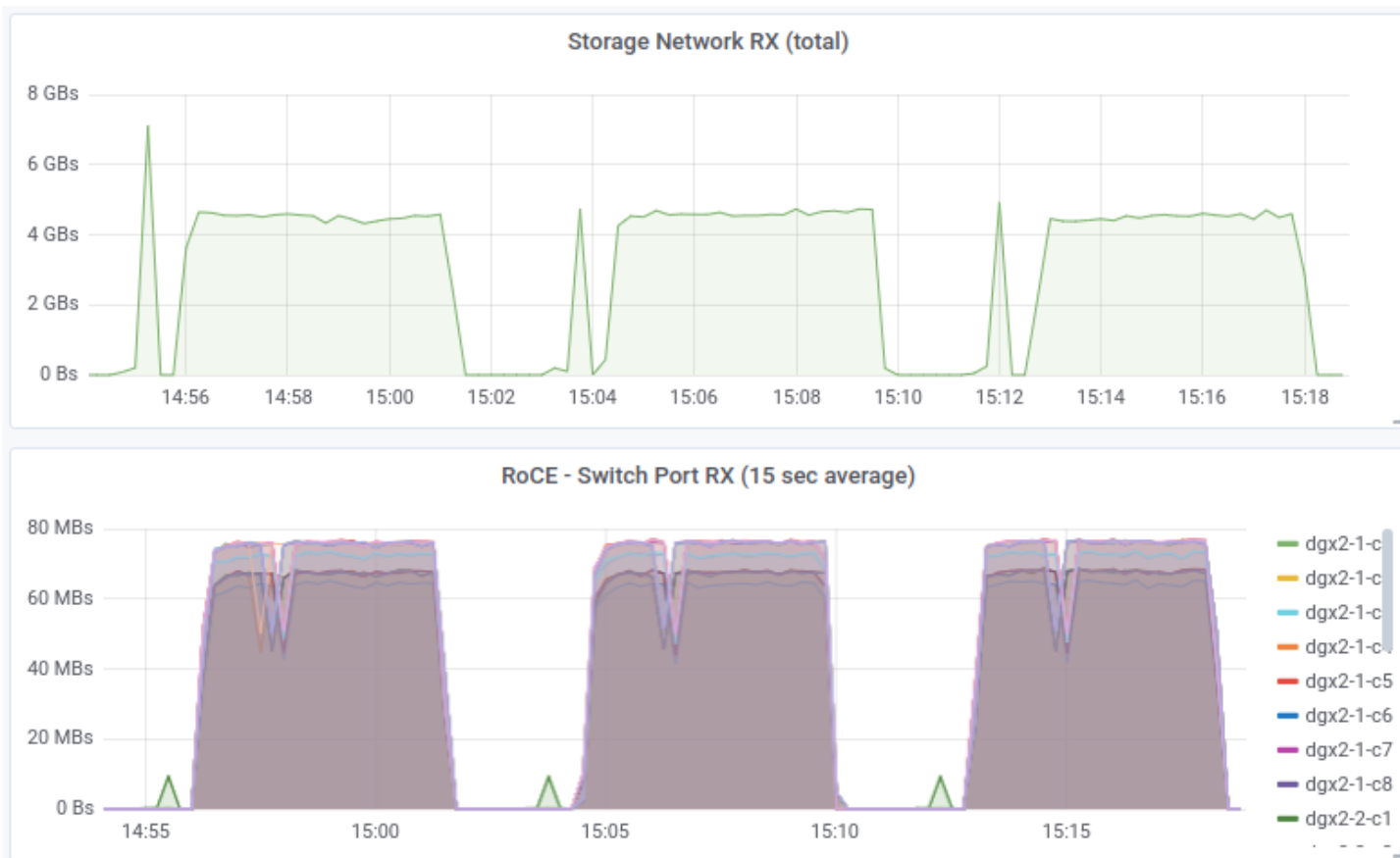


Figure 9: Network I/O during ResNet-50 training on 48 GPUs
(Note that the 24 RoCE switch ports have extreme bursts not captured in the above chart.)

MEASUREMENT OF NETWORK I/O BETWEEN DGX-2 SYSTEMS

Each DGX-2 system has eight 100 Gbps links that connect to other DGX-2 systems. These links are dedicated for the GPUs to communicate with each other when they are on different servers. These links connect to the Dell PowerSwitch 100 Gbps switches. When used with such Ethernet switches, RoCE (RDMA over Converged Ethernet) is used, which allows a GPU to communicate with a NIC directly across the PCIe bus, without involving the CPU.

The training optimization algorithm used for these benchmarks is synchronous and it requires all 48 GPUs to stop and exchange gradients before they can move on to the next batch. As shown in Figure 10, the RoCE network traffic is zero almost always but there are short bursts to 80 Gbps (per link per direction) every 0.39 seconds, which is the duration of each batch. There are 5 batches in this chart, spanning 2 seconds. The large peak at 0.55 sec corresponds to the MPI *all_reduce* call that aggregates the gradients. There is also a smaller peak at 0.37 seconds, the nature of which has not been investigated. Figure 11 shows the same data but is zoomed in to only one batch.

Among the models tested, VGG-16 used the highest amount of network bandwidth. As shown in Figure 12, ResNet-50 was observed to have a peak bandwidth of 60 Gbps (per link per direction).

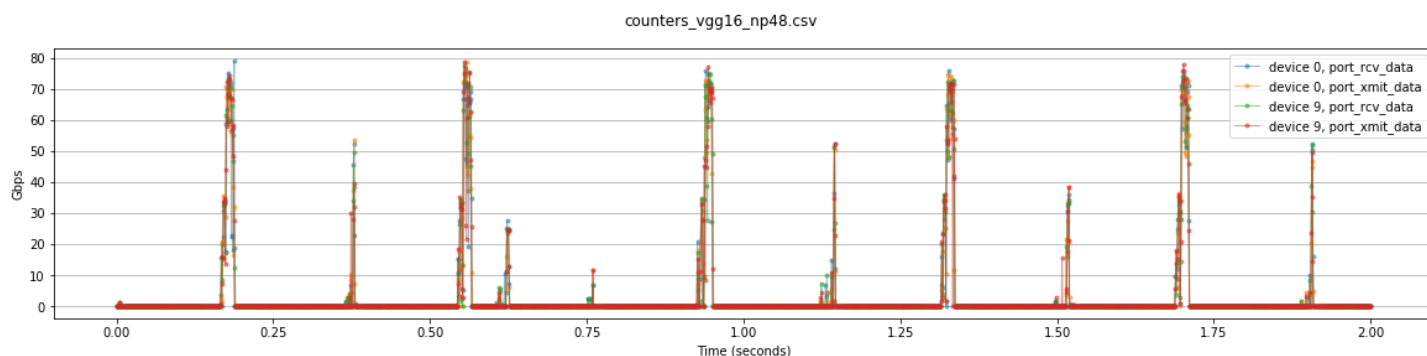


Figure 10: RoCE network traffic during VGG-16 training

Each batch has a duration of 0.39 seconds. There are 5 batches in this chart, spanning 2 seconds.

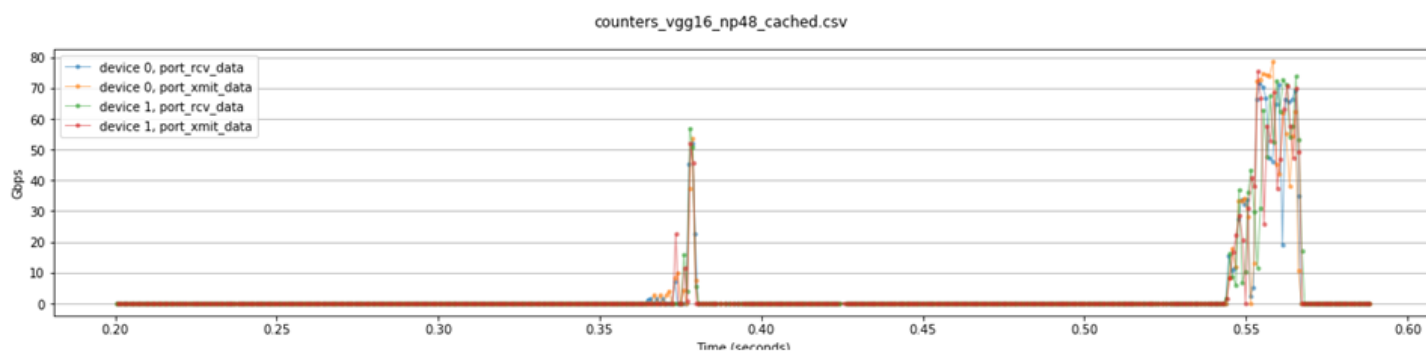


Figure 11: RoCE network traffic during VGG-16 (zoomed in to one batch)

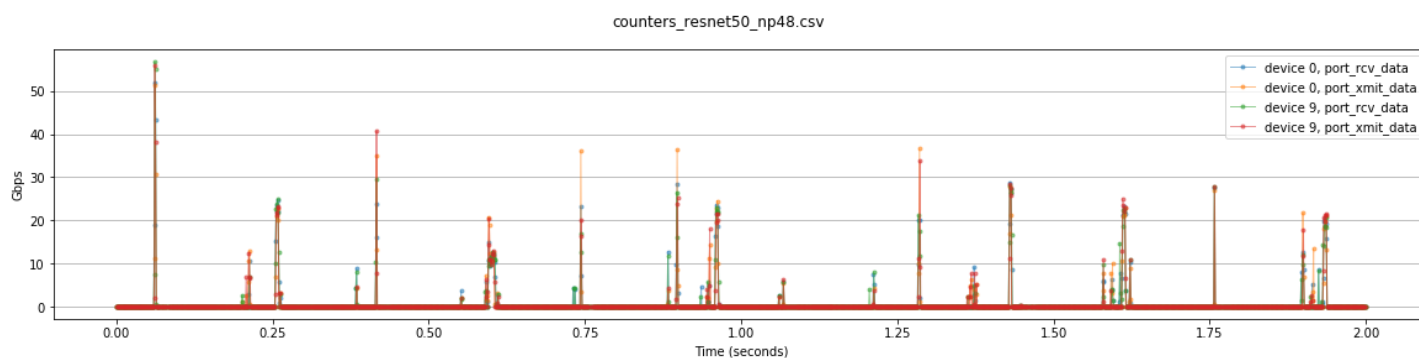


Figure 12: RoCE network traffic during ResNet-50 training

Note: The 15-second average link throughput is only 3 Gbps in Figure 10. Many monitoring tools sample network traffic at 10-30 second intervals. These tools are not able to capture the short but extremely high bursts. You must sample the traffic every 1-2 milliseconds to capture this behavior accurately.

To produce the charts above, we used the script `record_mlx_counters.py` in *AI Benchmark Util* to read the interface byte counter `/sys/class/infiniband/mlx5_0/ports/0/counters/port_xmit_data` approximately once per millisecond.

Note: The *storage* network traffic between the DGX-2 systems and Isilon is smooth, not bursty. This is due to pipelining, buffering, and read-ahead in the application.

UNDERSTANDING FILE CACHING

There are several caches related to NFS and Isilon storage.

Isilon Cache

An Isilon cluster has L1 and L2 cache which utilize the RAM on each Isilon node.

Linux Buffer Cache

A DGX-2 system, like other Linux-based hosts, has a buffer cache. The Linux buffer cache uses RAM that is otherwise unused to store recently used file blocks that were read from, or written to, local disks or NFS filesystems such as Isilon. There is generally no need to tune the Linux buffer cache and it is enabled by default on all modern versions of Linux. It will automatically grow as more data is read from disks (local and NFS) and it will be released when applications need additional RAM.

Linux NFS File System Cache

When enabled, the Linux NFS File System Cache uses the local disk to cache NFS files. In the solution described in this document (including all benchmarking), File System Cache is not used. In some cases, such as when using a slow NFS server, it may help to enable the File System Cache and point it to a local SSD. It is disabled by default. It can be enabled by settings the `fsc` mount option and starting `cachefilesd`.

UNDERSTANDING THE TRAINING PIPELINE

To tune the performance of the training pipeline, it is helpful to understand the key parts of the pipeline shown in Figure 13.

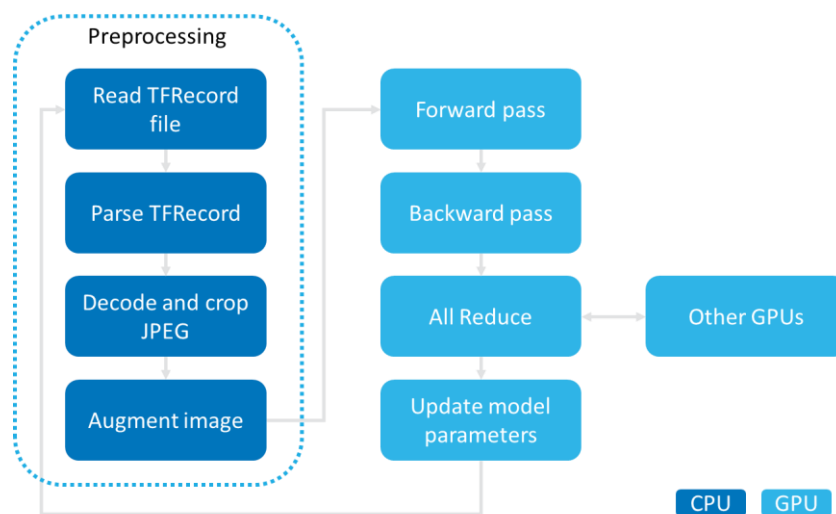


Figure 13: Training pipeline

1. Preprocessing

- (CPU) Read TFRecord files from disk. This may require NFS I/O to the storage system or it may be served by the Linux buffer cache in RAM.
- (CPU) Parse the TFRecord file into individual records and fields. The JPEG-encoded image is in one of these fields.
- (CPU) The JPEG is cropped and decoded. We now have an uncompressed RGB image. This is generally the most CPU-intensive step and can become a bottleneck in certain cases.
- (CPU) The cropped image is resized to 299x299 (Inception only) or 224x224 (all other models).
- (CPU) The image is randomly flipped horizontally or vertically.
- (CPU) If distortions are enabled, the image brightness is randomly adjusted. Note that distortions were disabled for all benchmarks described in this document.
- (CPU and GPU) The image is copied from the CPU RAM to the GPU RAM.

2. Forward and Backward Pass

- a. (GPU) Run the image through the model's forward pass (evaluate loss function) and backward pass (back propagation) to calculate the gradient.

3. Optimization

- a. (GPU) All GPUs across all nodes exchange and combine their gradients through the network using the All Reduce algorithm. In this solution, the communication is accelerated using NCCL and NVSwitch technology, allowing the GPUs to communicate through the Ethernet network, bypassing the CPU and PCIe buses.
- b. (GPU) The model parameters are updated based on the combined gradients and the optimization algorithm (gradient descent).
- c. Repeat until the desired accuracy (or another metric) is achieved.

All the steps above are done concurrently. For example, one CPU thread is reading a TFRecord file, while another is performing JPEG decoding on a file that was read several milliseconds prior and this occurs while the GPU is calculating gradients on images that were resized several milliseconds ago. Additionally, there is batching and buffering at various stages to optimize the performance. As you can see, the preprocessing pipeline is completely handled by the CPUs on the DGX-2 system.

NVIDIA COLLECTIVE COMMUNICATION LIBRARY (NCCL)

There are two significant types of network traffic when performing training. First, there is the NFS traffic required to read the images (TFRecord files). This uses standard TCP/IP on Ethernet. Second, the gradient (the derivative of the loss function with respect to each parameter) calculated on each GPU must be averaged with the gradient from all other GPUs and the resulting average gradient must be distributed to all GPUs. This is performed optimally using the MPI All Reduce algorithm. See Figure 14 below, which optimally calculates the sum of values from all nodes (top) and stores the sum on all nodes (bottom).

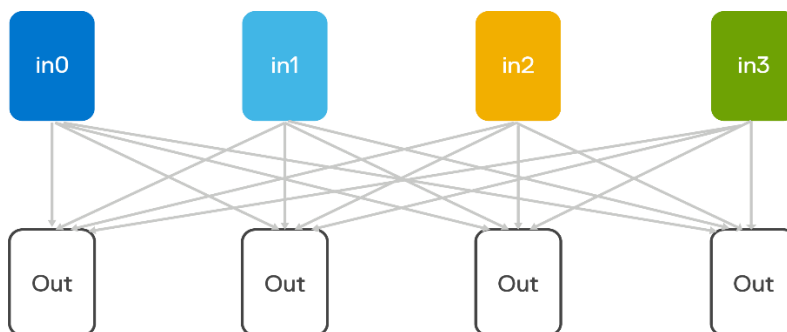


Figure 14: The All Reduce algorithm.

NCCL provides fast collectives (such as All Reduce) over multiple GPUs both within and across nodes. It supports a variety of interconnect technologies including PCIe, NVSwitch technology, InfiniBand Verbs, and IP sockets. NCCL also automatically patterns its communication strategy to match the system's underlying GPU interconnect topology. When configured properly on the DGX-2 system, NCCL allows GPUs in different nodes to communicate with each other through the PCIe switch, NIC, and the Ethernet switch, bypassing the CPU and the main system memory.

The `mpirun` parameters below will configure NCCL to use the RoCE-enabled NICs.

```
-x NCCL_DEBUG=INFO \
-x NCCL_IB_HCA=mlx5 \
-x NCCL_IB_SL=4 \
-x NCCL_IB_GID_INDEX=3 \
-x NCCL_NET_GDR_READ=1 \
-x NCCL_SOCKET_IFNAME=^docker0,lo \
```

To confirm that NCCL is optimally configured, carefully review the log messages of the training job and confirm that there are lines with "via NET/IB/x/GDRDMA". For example:

```

dgx2-1:25174:27350 [0] NCCL INFO NET/IB : Using [0]mlx5_6:1/RoCE [1]mlx5_8:1/RoCE
[2]mlx5_1:1/RoCE [3]mlx5_3:1/RoCE [4]mlx5_5:1/RoCE [5]mlx5_7:1/RoCE [6]mlx5_9:1/RoCE
[7]mlx5_0:1/RoCE [8]mlx5_2:1/RoCE [9]mlx5_4:1/RoCE ; OOB enp6s0:172.28.10.180<0>
dgx2-3:25000:27099 [8] NCCL INFO Ring 00 : 24 -> 40 [receive] via NET/IB/0/GDRDMA
dgx2-3:25000:27099 [8] NCCL INFO Ring 00 : 40 -> 24 [send] via NET/IB/0/GDRDMA

```

Storage-only performance

STORAGE NETWORK PERFORMANCE USING IPERF

The DGX-2 systems as described in the BOM and which were used for TensorFlow training benchmarks had a single dual-port storage NIC. To determine the storage network performance limits, iPerf was used to generate and measure TCP traffic between two DGX-2 systems using two 100 GbE storage network interfaces per server. This was found to provide a total throughput of only 110 Gbps. Since these two interfaces were on the same dual-port NIC, it was suspected that this reduced throughput was the result of the PCIe 3.0 x16 throughput limit of 126 Gbps.

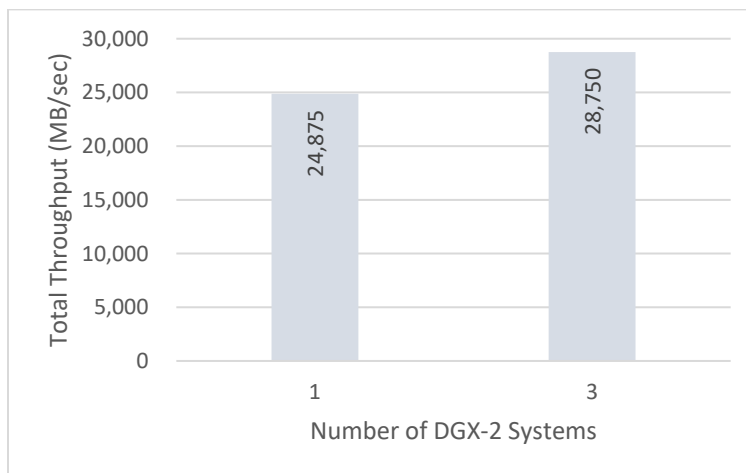
In light of this PCIe limit, for each DGX-2 system, we installed a second dual-port storage NIC and moved one network cable to the new NIC. The same iPerf test was repeated and it reported a throughput of 203 Gbps.

Refer to the iPerf section in the appendix for details on how iPerf was executed.

Note: All other results presented in this *Storage-only performance* section used DGX-2 systems with two dual-port storage NICs per server. All other results in this document used DGX-2 systems with one dual-port storage NIC per server.

STORAGE-ONLY PERFORMANCE USING FIO

To understand the limits of Isilon storage I/O in this environment, we used the common storage benchmark tool FIO to perform concurrent sequential reads from Isilon. A single DGX-2 system with two dual-port storage NICs was measured to read from Isilon at 24,875 MB/sec (199 Gbps), saturating both storage links. With three DGX-2 systems, we hit the throughput limit of the eight Isilon F800 nodes of about 28,750 MB/sec, shown in Figure 15. This is 4% slower than the rated capacity of 15 GB/sec per Isilon chassis under ideal workloads.

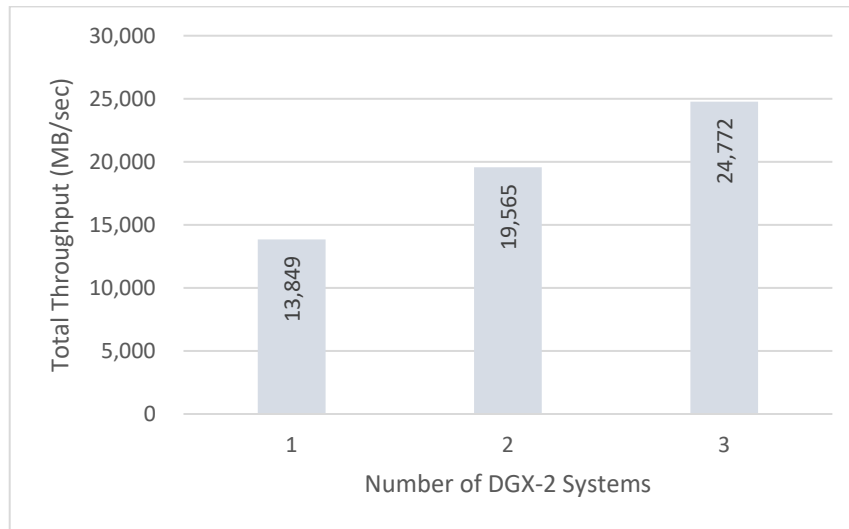


*Figure 15: FIO read performance from Isilon
(Each DGX-2 system has two dual-port storage NICs)*

Refer to the FIO section in the appendix for details on how FIO was executed.

STORAGE-ONLY PERFORMANCE USING TENSORFLOW

To understand the limits of Isilon when used with TensorFlow, a TensorFlow application was created (TensorFlow Storage Benchmark) that only reads the TFRecord files (the same ones that were used for training). No preprocessing nor GPU computation is performed. The only work performed is counting the number of bytes in each TFRecord. This application also has the option to synchronize all readers after each batch of records, forcing them to go at the same speed. This option was enabled to better simulate a DL or ML training workload. The result of this benchmark is shown in Figure 16.



*Figure 16: Synchronous reading from Isilon using TensorFlow Storage Benchmark
(Each DGX-2 system has two dual-port storage NICs)*

To determine the limitations of this benchmark methodology, we used the same Linux Cache technique used during the TensorFlow training benchmark. Specifically, we used the 148 GB dataset which the three DGX-2 systems were able to cache. We measured a total throughput of 108,529 MB/sec. Next, we used the larger 22 TB dataset, which required I/O to Isilon, and found that the throughput was reduced to 24,772 MB/sec.

To analyze this, we note that the workloads are identical except for the following aspects:

1. The DGX-2 system, like all servers, uses additional CPU when reading from NFS compared to reading from Linux cache.
2. There is significant storage network traffic from Isilon to the DGX-2 systems.
3. There is significant Isilon read I/O.

To investigate possible CPU limits on the DGX-2 system, we carefully analyzed the system metrics with one, two, and three DGX-2 systems running the benchmark with the 22 TB dataset (Figure 17 and Figure 18). Because all processes are forced to run at the same speed for this benchmark, the busiest CPU core is the best indicator of CPU limits. With one and two DGX-2 systems, the busiest CPU core was at 87% utilization, although some CPU cores had much less utilization. When this same benchmark was run with three DGX-2 systems, the busiest CPU core was only at 71% utilization, suggesting that the busiest CPU core could have been utilized an additional 16%. From this data, we can conclude that the bottleneck for the three DGX-2 systems was not the CPU (item 1 in the list above).

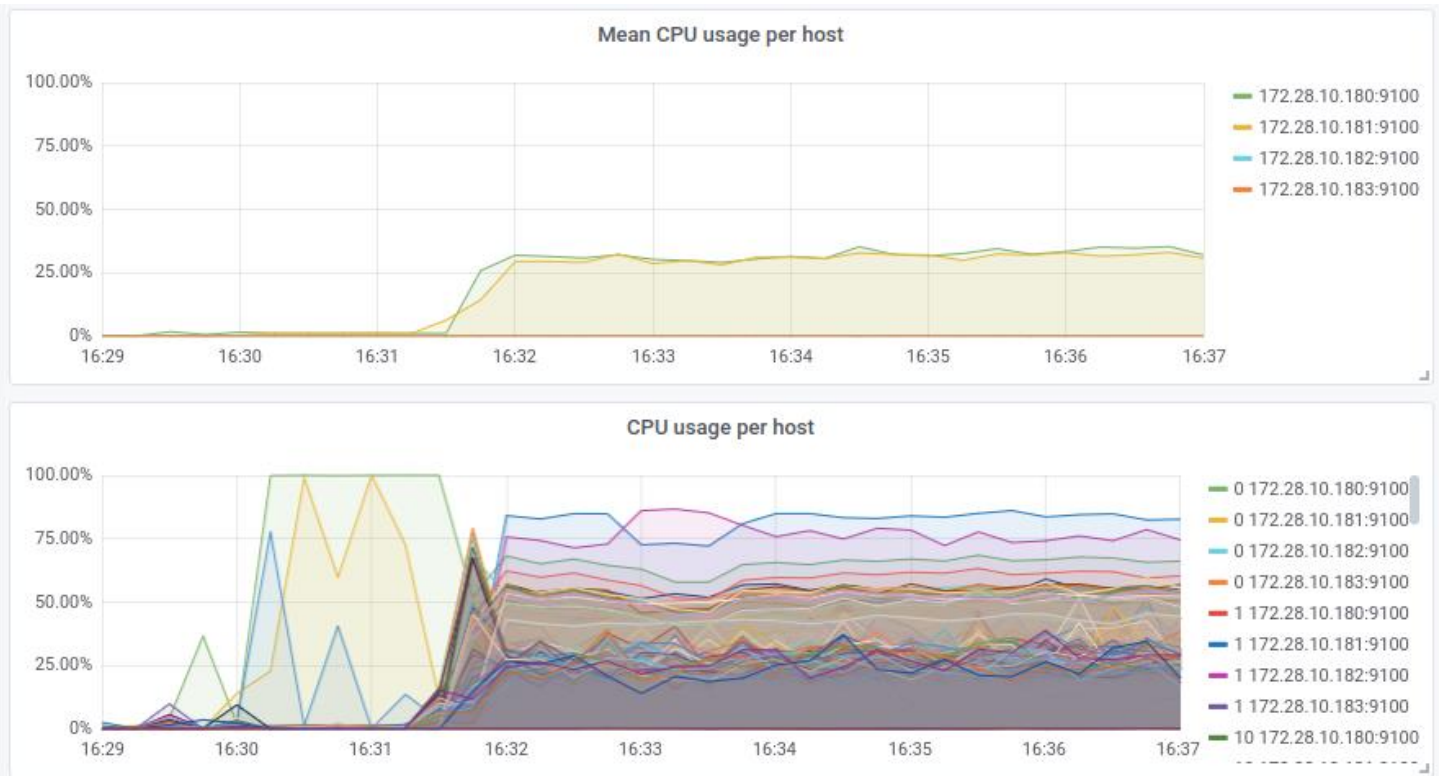


Figure 17: DGX-2 system CPU usage during TensorFlow Storage Benchmark with two DGX-2 systems.
The busiest core is at 87% utilization.

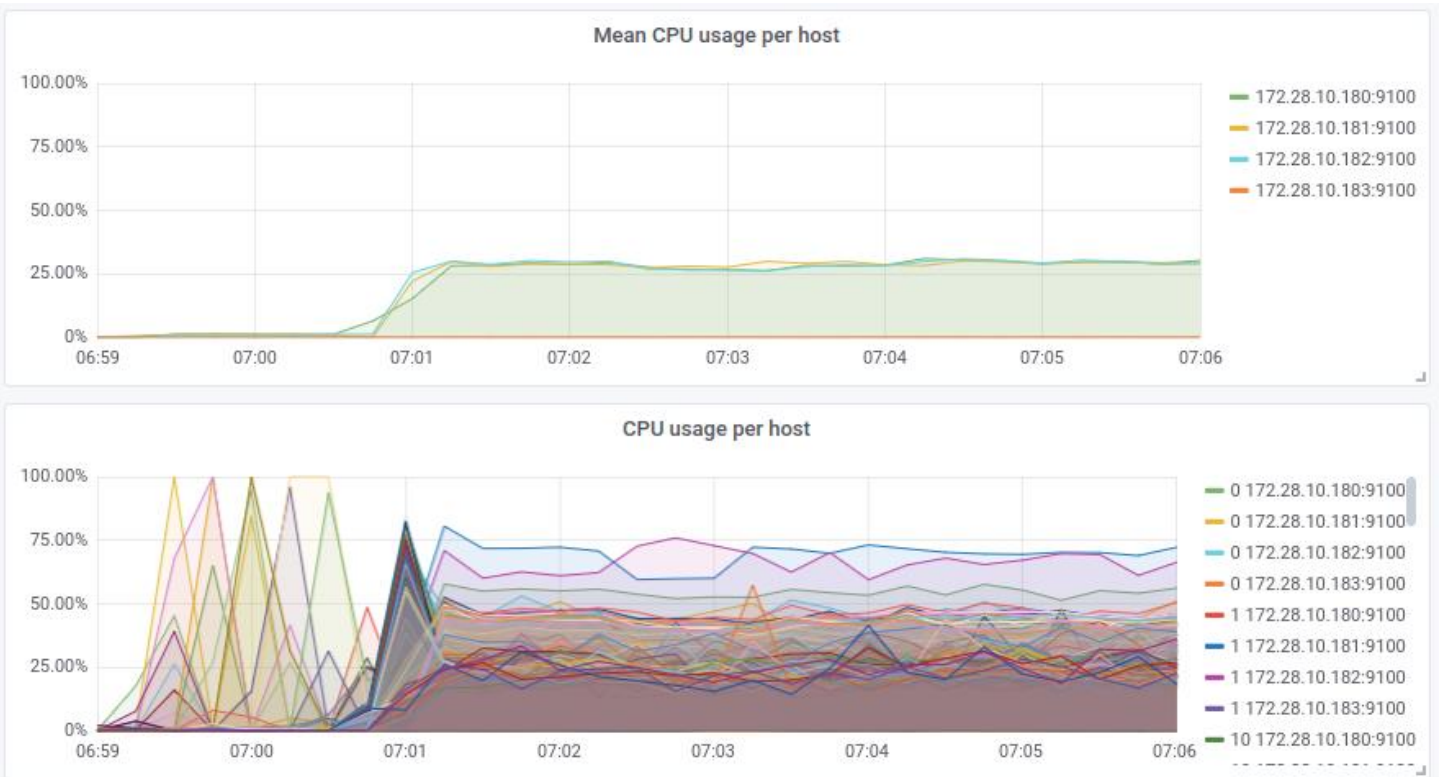


Figure 18: DGX-2 system CPU usage during TensorFlow Storage Benchmark with three DGX-2 systems.
The busiest core is at 71% utilization.

Next, we analyzed all network links in the solution, including DGX-2 storage interfaces, Isilon interfaces, and inter-switch links. As shown in Figure 19, all network links were below 50% capacity, eliminating item 2 in the list above as a possible bottleneck.

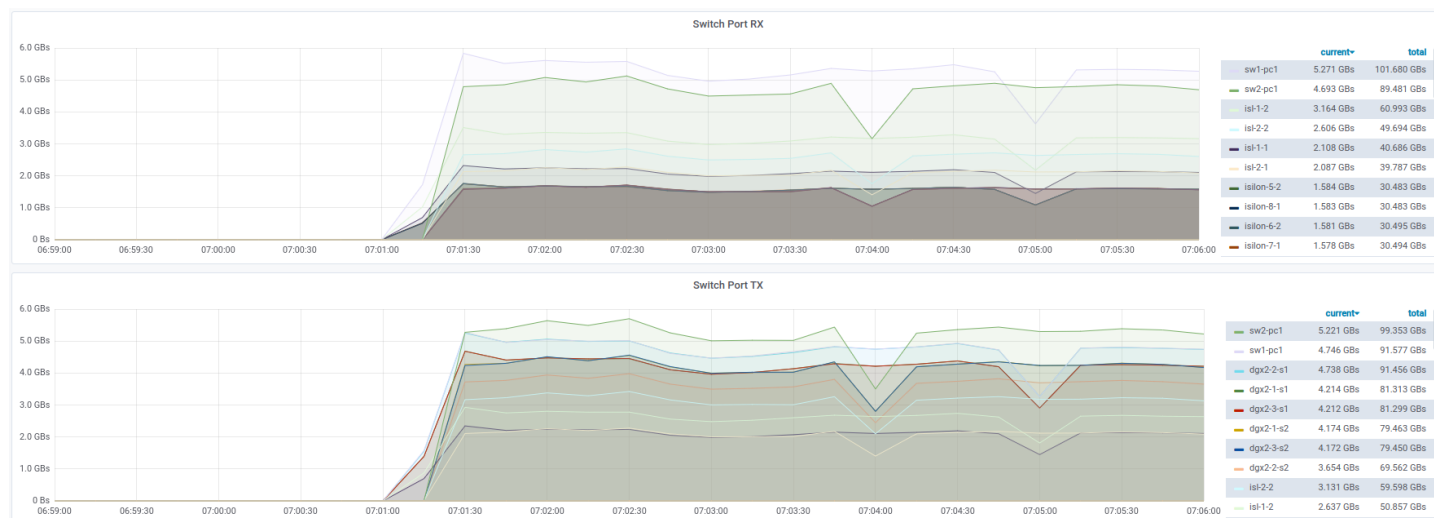


Figure 19: Switch port throughput during TensorFlow Storage Benchmark with 3 DGX-2 systems. Port capacities are 40 Gbps (5 GB/s) for Isilon, 200 Gbps (25 GB/s) for the port channels (swx-pc1), and 100 Gbps (12.5 GB/s) for all other ports. All ports are less than 50% utilized.

This leaves only item 3 as a possible bottleneck in the system and we can conclude that the throughput performance for this benchmark using two Isilon F800 chassis (eight nodes) is 24,772 MB/sec. This is 14% slower than the 28,750 MB/sec obtained with FIO. The key difference between these two benchmarks is that the threads of FIO are independent and Isilon can read the different files at different speeds in a way that optimizes the total throughput. In contrast, the TensorFlow Storage Benchmark will effectively pause faster threads to allow slower threads to catch up.

As Isilon has been demonstrated to scale to 252 nodes, additional throughput can be obtained simply by adding Isilon nodes.

Refer to Appendix – Isilon performance testing with TensorFlow for details on how to execute this benchmark.

Note: This data suggests that for the benchmarks with one and two DGX-2 system, the bottleneck was likely the DGX-2 system CPU. However, an exhaustive analysis of this particular synthetic workload is beyond the scope of this document.

Solution sizing guidance

DL workloads vary significantly with respect to the demand for compute, memory, disk, and I/O profiles, often by orders of magnitude. Sizing guidance for the GPU quantity and configuration of Isilon nodes can only be provided when these resource requirements are known in advance. That said, it's usually beneficial to have a few data points on the ratio of GPUs per Isilon node for common image classification benchmarks.

The results in Table 3 show a few such data points:

Storage Performance Demanded	Benchmark	Minimum Required Storage Throughput per Tesla V100 GPU (MB/sec/GPU)	Maximum Tesla V100 GPUs per Isilon Node	
			F800	H500
Low	Training, Small Images, ResNet-50	100	30 ²	7 ³
Medium	Inference, Small Images, ResNet-50	160	22 ⁴	5
High	Training, Large Images, Inception-v4	300	10	2

Table 3: Sizing consideration based on benchmarked workloads

To illustrate, training of ResNet-50 with small images and 48 GPUs would need a storage system that can read $48 * 90 = 4320$ MB/sec which can be handled by $48 / 30 = 1.6$ Isilon F800 nodes.

Note: For DL workloads and datasets that require a greater ratio of capacity to throughput or capacity to price than what the F800 provides, the Isilon H500 hybrid storage platform may be a good choice. Per node, the H500 delivers about 25-33% of the throughput of the F800 for sequential reads. A cluster may contain exclusively H500 nodes, two tiers including F800 and H500 nodes, or a variety of other combinations to suit your specific workload and datasets. In Table 3, the H500 sizing information is based on general NFS benchmarks. *The H500 has not been tested at scale with the specific DL workloads described in this document.*

The above data points do not account for node failure, drive failure, network failure, administrative jobs, or any other concurrent workloads supported by Isilon including other data preparation tasks. The ideal ratio of number of GPUs per Isilon node will vary from the above data points based on several factors:

- DL algorithms are diverse and don't necessarily have the same infrastructure demands as the benchmarks listed above.
- Characteristics and size of the data sets will be different from the data sets described in this document and used to generate the above data points.
- Accounting for node/drive/network failures, admin jobs or other workloads accessing Isilon simultaneously.
- The data points above are *minimums*. Production systems should have extra storage throughput and capacity, and the general recommendation is to add two nodes to the minimum required number of Isilon nodes.

An understanding of the I/O throughput demanded per GPU for the specific workload and the total storage capacity requirements can help provide better guidance on Isilon node count and configuration. It is recommended to reach out to the Dell EMC account and SME teams to provide this guidance for the specific DL workload, throughput, and storage requirements.

Conclusions

This document presented a high-performance architecture for DL by combining NVIDIA DGX-2 systems with Tesla V100 GPUs, Dell EMC PowerSwitch S5232F-ON switches, and Dell EMC Isilon F800 all-flash NAS storage. We discussed key features of Isilon that make it a powerful persistent storage for DL solutions. This new reference architecture extends the commitment that Dell Technologies and NVIDIA made to making AI simple and accessible to every organization with our unmatched set of joint offerings. Together we offer our customers informed choice and flexibility in how they deploy high-performance DL at scale.

To validate this architecture, we ran several image classification benchmarks and reported system performance based on the rate of images processed and throughput profile of I/O to disk. We also monitored and reported the CPU, GPU utilization, and memory statistics that demonstrated that the server, GPU and memory resources were fully utilized

² The Isilon performance limit for training is assumed to equal the performance achieved with TensorFlow Storage Benchmark in synchronous mode described in Storage-only performance using TensorFlow.

³ Isilon H500 performance for these sequential workloads is approximated to be 25% of F800 performance.

⁴ The Isilon performance limit for inference is assumed to equal the performance achieved with FIO read benchmarking described in Storage-only performance using FIO.

while I/O was not fully saturated. Throughout the benchmarks we validated that the Isilon all-flash F800 storage was able to keep pace and linearly scale performance with DGX-2 systems.

It is important to point out that DL algorithms have a diverse set of requirements with various compute, memory, I/O, and disk capacity profiles. That said, the architecture and the performance data points presented in this whitepaper can be utilized as the starting point for building DL solutions tailored to varied sets of resource requirements. More importantly, all the components of this architecture are linearly scalable and can be independently expanded to provide DL solutions that can manage tens of PBs of data.

While the solution presented here provides several performance data points and speaks to the effectiveness of Isilon in handling large scale DL workloads, there are several other operational benefits of persisting data for DL on Isilon:

- The ability to run AI in-place on data using multi-protocol access
- Enterprise grade features out-of-box
- Seamlessly tier to more cost-effective nodes
- Scale up to 58 PB per cluster

In summary, Isilon-based DL solutions deliver the capacity, performance and high concurrency to eliminate the I/O storage bottlenecks for AI. This provides a rock-solid foundation for large scale, enterprise-grade DL solutions with a future proof scale-out architecture that meets your AI needs of today and scales for the future.

Acknowledgements

The authors gratefully acknowledge the contributions that were made to this document by our colleagues from NVIDIA: Jacci Cenci, Joe Handzik, Darrin Johnson, and Robert Sohigian. The authors would also like to acknowledge the contributions of key Dell EMC team members Frances Hu, Mark Orth, and Sasha Paegle.

Appendix – System configuration

ISILON

Configuration

The Isilon cluster configuration is simple yet it provides excellent performance.

There are two subnets, each with one pool. One subnet is for 40gige-1 and 40gige-2 on all nodes and the other is for mgmt-1 which is connected only to two nodes in our lab. In total, sixteen (16) 40 GbE ports are used in the Isilon cluster.

isilon-1# isi network interfaces list					
LNN	Name	Status	Owners	IP Addresses	
1	40gige-1	Up	groupnet0.data.pool1	10.200.10.151	
1	40gige-2	Up	groupnet0.data.pool1	10.200.10.153	
1	mgmt-1	Up	groupnet0.subnet0.pool0	172.28.10.151	
2	40gige-1	Up	groupnet0.data.pool1	10.200.10.152	
2	40gige-2	Up	groupnet0.data.pool1	10.200.10.161	
2	mgmt-1	No Carrier	-	-	
3	40gige-1	Up	groupnet0.data.pool1	10.200.10.165	
3	40gige-2	Up	groupnet0.data.pool1	10.200.10.159	
3	mgmt-1	No Carrier	-	-	
4	40gige-1	Up	groupnet0.data.pool1	10.200.10.154	
4	40gige-2	Up	groupnet0.data.pool1	10.200.10.163	
4	mgmt-1	No Carrier	-	-	
5	40gige-1	Up	groupnet0.data.pool1	10.200.10.155	
5	40gige-2	Up	groupnet0.data.pool1	10.200.10.166	
5	mgmt-1	No Carrier	-	-	
6	40gige-1	Up	groupnet0.data.pool1	10.200.10.156	
6	40gige-2	Up	groupnet0.data.pool1	10.200.10.164	
6	mgmt-1	No Carrier	-	-	
7	40gige-1	Up	groupnet0.data.pool1	10.200.10.157	
7	40gige-2	Up	groupnet0.data.pool1	10.200.10.162	
7	mgmt-1	No Carrier	-	-	
8	40gige-1	Up	groupnet0.data.pool1	10.200.10.158	
8	40gige-2	Up	groupnet0.data.pool1	10.200.10.160	
8	mgmt-1	Up	groupnet0.subnet0.pool0	172.28.10.152	

Total: 24

To have high availability in the event of an Isilon node failure, using the dynamic IP allocation method is recommended. This will ensure that all IP addresses in the pool are always available. Refer to the [OneFS Best Practices](#) document for details.

Configuring automatic storage tiering

This section explains one method of configuring storage tiering for a typical DL workload. The various parameters should be changed according to the expected workload.

1. Build an Isilon cluster with two or more node types. For example, F800 all-flash nodes for a hot tier and H500 hybrid nodes for a cold tier.
2. Obtain and install the Isilon SmartPools license.
3. Enable Isilon access time tracking with a precision of one day. In the web administration interface, click File System → File System Settings. Alternatively, enter the following in the Isilon CLI:





```
isilon-1# sysctl efs.bam.atime_enabled=1
isilon-1# sysctl efs.bam.atime_grace_period=86400000
```





4. Create a tier named *hot-tier* that includes the F800 node pool. Create a tier named *cold-tier* that includes the H500 node pool.

Storage Pools

Summary
File Pool Policies
SmartPools
CloudPools
SmartPools Settings
CloudPools Settings

Tiers & Node Pools + Create a Tier

Name	State	Nodes	Requested Protection	SSD/L3	HDD % Used	SSD % Used	Actions
 cold-tier	Good	5-8	--	L3 Cache	0.1%	--	View / Edit More
 h500_60tb_3.2tb-ssd_128gb	Good	5-8	+2d:1n	L3 Cache	0.1%	--	View / Edit More
 hot-tier	Good	1-4	--	Has SSDs	0.0%	0.0%	View / Edit More
 f800_48tb-ssd_256gb	Good	1-4	+2d:1n	Has SSDs	0.0%	0.0%	View / Edit More

 = Tier
 = Node Pool
 = Manual Node Pool
 = Unprovisioned Node

- Edit the default file pool policy to change the storage target to *hot-tier*. This will ensure that all files are placed on

View Default Policy Details Help ?

* = Required field

Policy Information

Policy Name
Default Policy

Description
This policy applies to all files not selected by higher-priority policies.

Select Files to Manage

File Matching Criteria
Matches all files not already matched by any other policies

Apply SmartPools Actions to Selected Files

Storage Settings

Move To Storage Pool or Tier
Storage Target: hot-tier (tier)
Use SSDs for data and metadata (Requires the most SSD space)

Move Snapshots to Storage Pool or Tier
Snapshot Storage Target: hot-tier (tier)
Use SSDs for data and metadata (Requires the most SSD space)

Requested Protection
Using requested protection of the node pool or tier (Suggested)

I/O Optimization Settings

Write Performance
SmartCache is enabled

Data Access Pattern
Optimized for concurrent access

the F800 nodes unless another file pool policy applies that overrides the storage target.

6. Create a new file pool policy to move all files that have not been accessed for 30 days to the H500 (cold) tier.

View File Pool Policy Details
[Help](#)

* = Required field

Description

CloudPools State
No access

CloudPools State Details
Policy has no CloudPools actions

* **Policy Name**
Idle data to cold tier

Description
No value

Select Files to Manage

* **File Matching Criteria**
IF
Accessed is older than 1 month ago

Apply SmartPools Actions to Selected Files

Storage Settings

Move To Storage Pool or Tier
Storage Target: cold-tier (tier)
Use SSDs for metadata read acceleration (Recommended)

Move Snapshots to Storage Pool or Tier
Snapshot Storage Target: cold-tier (tier)
Use SSDs for metadata read acceleration (Recommended)

Requested Protection

[I/O Optimization Settings](#)

Testing automatic storage tiering

1. Start the Isilon SmartPools job and wait for it to complete. This will apply the above file pool policy and move all files to the F800 nodes (assuming that all files have been accessed within the last 30 days).

```
isilon-1# isi job jobs start SmartPools
isilon-1# isi status
```

2. Use `isi get` to confirm that a file is stored in the hot tier.
3. The first number in each element of the inode list (1, 2, and 3 below) is the Isilon node number that contains blocks of the file. For example:

```
isilon-1# isi get -D /ifs/data/imagenet-scratch/tfrecords/train-00000-of-01024
* IFS inode: [ 1,0,1606656:512, 2,2,1372672:512, 3,0,1338880:512 ]
* Disk pools:          policy hot-tier(5) -> data target f800_48tb-
                        ssd_256gb:2(2), metadata target f800_48tb-ssd_256gb:2(2)
```

4. Use the `ls` command to view the access time.

```
isilon-1# ls -lu /ifs/data/imagenet-scratch/tfrecords/train-00000-of-01024
-rwx----- 1 1000 1000 762460160 Jan 16 19:32 train-00000-of-01024
```

5. Instead of waiting for 30 days, you may manually update the access time of the files using the `touch` command.

```
isilon-1# touch -a -d '2018-01-01T00:00:00' \
/ifs/data/imagenet-scratch/tfrecords/*
isilon-1# ls -lu /ifs/data/imagenet-scratch/tfrecords/train-00000-of-01024
-rwx----- 1 1000 1000 762460160 Jan 1 2018 train-00000-of-01024
```

6. Start the Isilon SmartPools job and wait for it to complete. This will apply the above file pool policy and move the files to the H500 nodes.

```
isilon-1# isi job jobs start SmartPools
isilon-1# isi status
```

7. Use the *isi get* command to confirm that the file is stored in the cold tier.

```
isilon-1# isi get -D /ifs/data/imagenet-scratch/tfrecords/train-00000-of-01024
* IFS inode: [ 4,0,1061376:512, 5,1,1145344:512, 6,3,914944:512 ]
* Disk pools:          policy cold-tier(9) -> data target h500_60tb_3.2tb-
ssd_128gb:15(15), metadata target h500_60tb_3.2tb-ssd_128gb:15(15)
```

8. Run the training benchmark as described in the previous section.
9. Be sure to run at least four epochs so that all TFRecords are accessed. Monitor the Isilon cluster to ensure that disk activity occurs only on the H500 nodes. Note that since the files will be read, this will update the access time to the current time. When the SmartPools job runs next, these files will be moved back to the F800 tier. By default, the SmartPool job runs daily at 22:00.
10. Start the Isilon SmartPools job and wait for it to complete. This will apply the above file pool policy and move the files to the F800 nodes.

```
isilon-1# isi job jobs start SmartPools
isilon-1# isi status
```

11. Use the *isi get* command to confirm that the file is stored in the hot tier.

```
isilon-1# isi get -D /ifs/data/imagenet-scratch/tfrecords/train-00000-of-01024
* IFS inode: [ 1,0,1606656:512, 2,2,1372672:512, 3,0,1338880:512 ]
* Disk pools:          policy hot-tier(5) -> data target f800_48tb-
ssd_256gb:2(2), metadata target f800_48tb-ssd_256gb:2(2)
```

DELL EMC POWERSWITCH S5232F-ON DATA SWITCHES

There are several important items to configure on the data switches.

1. Isilon ports must be set to 40 GbE speed.
2. Each of the eight Isilon nodes has two 40 GbE ports labeled 40gige-1 and 40gige-2. To ensure high-availability, the two ports on each Isilon node should connect to different switches.
3. An aggregated port channel consisting of two 100 GbE links connects the two data switches. This is referred to as the inter-switch link. This carries only NFS traffic and is on VLAN 1.
4. RDMA over Converted Ethernet (RoCE) traffic is on VLAN 55. This traffic will never travel through the inter-switch link.
5. Ethernet MTU is set to the maximum value of 9216. This value must be larger than the IP MTU (9000) since it must account for Ethernet headers.
6. SNMP is enabled to allow traffic to be monitored by Prometheus. All active ports should have useful and concise descriptions so that time series are labeled properly.

The switch configuration is detailed in Appendix – Switch configuration.

NVIDIA DGX-2 SYSTEM

The DGX-2 system as tested has a total of ten 100 GbE network ports. Eight of these are associated with specific pairs of GPUs and they are used for high speed low latency RDMA traffic between GPUs across the network. The other two 100 GbE network ports are used for NFS traffic between the DGX-2 systems and Isilon.

Note: DL benchmarking was performed with a single dual-port ConnectX5 NIC used for both storage interfaces.

Follow the DGX-2 system administrator's manual to configure the NICs to use Ethernet instead of InfiniBand. Use the commands below to confirm that they are configured correctly.

```
dgxuser@dgx2-1:~$ ibv_devinfo
hca_id: mlx5_8
transport:          InfiniBand (0)
```

```

fw_ver:          16.25.1020
node_guid:       9803:9b03:002f:0002
sys_image_guid:  9803:9b03:002f:0002
vendor_id:       0x02c9
vendor_part_id:  4119
hw_ver:          0x0
board_id:        MT_0000000010
phys_port_cnt:   1
Device ports:
  port: 1
    state:        PORT_ACTIVE (4)
    max_mtu:      4096 (5)
    active_mtu:   4096 (5)
    sm_lid:       0
    port_lid:     0
    port_lmc:     0x00
    link_layer:   Ethernet

```

...

The DGX-2 system uses netplan to configure networking. Netplan is configured to bond the two storage interfaces using Adaptive Load Balancing, which requires no switch configuration and provides excellent performance. MTU of 9000 is used for all 100 GbE interfaces. Setting MTU with netplan requires specifying the MAC address of each interface.

Below is an example of the `/etc/netplan/01-netcfg.yaml` file.

```

network:
  version: 2
  renderer: networkd
  ethernets:
    # management
    enp6s0:
      addresses: [ 172.28.10.180/26 ]
      gateway4: 172.28.10.130
      nameservers:
        addresses:
          - "8.8.8.8"
    # storage 1
    enp134s0f0:
      dhcp4: no
    # storage 2
    enp134s0f1:
      dhcp4: no
    # cluster 1
    enp53s0:
      addresses: [10.200.11.11/24]
      match:
        macaddress: 98:03:9b:33:d0:12
      mtu: 9000
    # cluster 2
    enp58s0:
      addresses: [10.200.12.11/24]
      match:
        macaddress: 98:03:9b:2f:49:18
      mtu: 9000
    # cluster 3
    enp88s0:
      addresses: [10.200.13.11/24]
      match:
        macaddress: 98:03:9b:2e:fc:aa
      mtu: 9000
    # cluster 4
    enp93s0:

```

```

    addresses: [10.200.14.11/24]
    match:
      macaddress: 98:03:9b:2e:fe:ce
    mtu: 9000
# cluster 5
enp184s0:
  addresses: [10.200.15.11/24]
  match:
    macaddress: 98:03:9b:2f:00:02
  mtu: 9000
# cluster 6
enp189s0:
  addresses: [10.200.16.11/24]
  match:
    macaddress: 98:03:9b:2f:02:6e
  mtu: 9000
# cluster 7
enp225s0:
  addresses: [10.200.17.11/24]
  match:
    macaddress: 98:03:9b:32:f5:32
  mtu: 9000
# cluster 8
enp230s0:
  addresses: [10.200.18.11/24]
  match:
    macaddress: 98:03:9b:2f:49:84
  mtu: 9000
bonds:
  bond0:
    addresses: [10.200.10.11/24]
    interfaces:
      - enp134s0f0
      - enp134s0f1
    parameters:
      mode: balance-alb
      primary: enp134s0f0
      mii-monitor-interval: 10
    mtu: 9000

```

Below is an example of the file */etc/hosts*.

```

127.0.0.1    localhost
127.0.1.1    dgx2-1
172.28.10.180 dgx2-1
172.28.10.181 dgx2-2
172.28.10.182 dgx2-3

```

INSTALL AI BENCHMARK UTILITIES

Throughout this document, we use several Bash and Python scripts that are available at [AI Benchmark Util](#). These should be installed on all DGX-2 systems.

```

cd /mnt/isilon/data
git clone https://github.com/claudiofahey/ai-benchmark-util
cd ai-benchmark-util
sudo apt install python3-pip
pip3 install setuptools
pip3 install --requirement requirements.txt

```

ISILON VOLUME MOUNTING

Isilon is used for two types of file storage. First, it is used for scripts, binaries and logs. This requires low bandwidth and must support NFS locks for consistency and proper visibility of changes. This generally uses the default mount options and is mounted with:

```
mount -t nfs 10.200.10.151:/ifs /mnt/isilon
```

Next, there is the data that will be read or written at high speed. To ensure an even balance of traffic across both 40 Gbps interfaces on each Isilon node and across two interfaces on each DGX-2 system, we'll want to carefully create several mounts explicitly to the IP addresses of several Isilon nodes.

As an example, the commands below can be run on each DGX-2 system to mount to each of the 16 Isilon interfaces.

```
mount -t nfs 10.200.10.151:/ifs \
rsize=524288,wsiz=524288,nolock /mnt/isilon1
mount -t nfs 10.200.10.152:/ifs \
rsize=524288,wsiz=524288,nolock /mnt/isilon2
...
mount -t nfs 10.200.10.166:/ifs \
rsize=524288,wsiz=524288,nolock /mnt/isilon16
```

Note that since training is purely a read workload, it is safe to add the `nolock` parameter to the mounts that contain the input data. In some cases, this can improve performance.

If you are using the recommend dynamic IP allocation policy in the Isilon IP address pool, all IP addresses will remain accessible, even in the event of an Isilon node failure.

If you are expecting to perform more than 2 GB/sec of I/O from a single DGX-2 system, you should ensure that the application uses multiple mount points to multiple Isilon nodes. Ideally, each DGX-2 system should use all 16 mount points evenly. This can be easily accomplished with a script like `round_robin_mpi.py` which uses the rank of the MPI process to select a mount point.

As a simple but less effective alternative, you may mount `/mnt/isilon1` to a different Isilon interface on each DGX-2 system, but this may result in a sub-optimal balance of traffic. Also, it has been observed that a single NFS mount can read about 2500 MB/sec which is only half of the 40 Gbps front-end Ethernet links on the Isilon nodes. Therefore, it is best to have at least two mounts per Isilon interface.

Note that different mounts do not share the Linux buffer cache. If your dataset is small enough to fit in the DGX-2 system RAM, consider using fewer mounts to allow your entire dataset to be cached.

For all benchmarks presented in this document, the script `mount_isilon.py` in *AI Benchmark Util* was used to automate the drive mounts.

Appendix – Benchmark setup

CREATING THE IMAGENET TFRECORD DATASETS

To run the TensorFlow Benchmarks suite, the standard 148 GB ImageNet TFRecord dataset was created based on the documentation at <https://github.com/tensorflow/models/tree/master/research/inception#getting-started>.

To create the 22.2 TB dataset, consisting of 150 copies of the above, the script `expand_tfrecords.sh` was used.

OBTAIN THE TENSORFLOW BENCHMARKS

The TensorFlow Benchmark suite can be obtained from the following Git repository.

```
cd /mnt/isilon/data
git clone https://github.com/claudiofahey/benchmarks tensorflow-benchmarks
cd tensorflow-benchmarks
git checkout 31ea13f
```

Note that the commit above differs from the official repository in only one significant way. It removes an unnecessary file name wildcard globbing step which has a huge performance impact when the number of TFRecord files exceeds 10,000. See <https://github.com/claudiofahey/benchmarks/commit/31ea13f>.

START TENSORFLOW CONTAINERS

In a basic bare-metal deployment of TensorFlow and MPI, all software must be installed on each node. MPI then uses SSH to connect to each node to start the TensorFlow application processes.

In the world of Docker containers, this becomes a bit more complex but significantly easier to manage dependencies with. On each DGX-2 system, a single Docker container is launched which has an SSH daemon that listens on the custom port 2222. This Docker container also has TensorFlow, OpenMPI and NVIDIA libraries and tools. We can then *docker exec* the *mpirun* command on one of these containers and MPI will connect to the Docker containers on all other DGX-2 systems via SSH on port 2222.

First, a custom Docker image is created using the following *Dockerfile*.

```
FROM nvcr.io/nvidia/tensorflow:19.09-py3

MAINTAINER Claudio Fahey <Claudio.Fahey@dell.com>

# Install SSH and various utilities.
RUN apt-get update && apt-get install -y --no-install-recommends \
    openssh-client \
    openssh-server \
    lsof \
    && \
    rm -rf /var/lib/apt/lists/*

# Configure SSHD for MPI.
RUN mkdir -p /var/run/sshd && \
    mkdir -p /root/.ssh && \
    echo "StrictHostKeyChecking no" >> /etc/ssh/ssh_config && \
    echo "UserKnownHostsFile /dev/null" >> /etc/ssh/ssh_config && \
    sed -i 's/^#*Port 22/Port 2222/' /etc/ssh/sshd_config && \
    echo "HOST *" >> /root/.ssh/config && \
    echo "PORT 2222" >> /root/.ssh/config && \
    mkdir -p /root/.ssh && \
    ssh-keygen -t rsa -b 4096 -f /root/.ssh/id_rsa -N "" && \
    cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys && \
    chmod 700 /root/.ssh && \
    chmod 600 /root/.ssh/*

# Install Python libraries.
COPY requirements.txt /tmp/requirements.txt
RUN pip install --requirement /tmp/requirements.txt

WORKDIR /scripts

EXPOSE 2222
```

As you can see, this *Dockerfile* is based on the [NVIDIA GPU Cloud \(NGC\) TensorFlow image](#).

Run the following command to build the Docker image. Replace user with your NGC ID, Docker ID, or *host:port* if you are using an on-premise container registry.

```
docker build -t user/tensorflow:19.09-py3-custom.
```

Note that during the build process, a new RSA key pair is randomly generated and stored in the image. This key pair allows containers running this image to SSH into each other. Although this is convenient for a lab environment, a production environment should never store private keys in an image.

Next, you must push this image to a Docker container registry so that it can be pulled from all other DGX-2 systems. You can use an NGC private repository, Docker Hub, or your own private on-premise container registry. To run an unsecure on-premise registry, refer to <https://docs.docker.com/registry/> and <https://docs.docker.com/registry/insecure/>.

Once logged in to your container registry, run the following command to upload the container.

```
docker push user/tensorflow:18.09-py3-custom.
```

You are now ready to start the containers on all DGX-2 systems. Repeat this command for each DGX-2 system, replacing host with the server name.

```
ssh host \  
nvidia-docker \  
run \  
--rm \  
--detach \  
--privileged \  
-v /mnt:/mnt \  
--network=host \  
--shm-size=1g \  
--ulimit memlock=-1 \  
--ulimit stack=67108864 \  
--name tf \  
user/tensorflow:19.09-py3-custom \  
bash -c \  
"/usr/sbin/sshd ; sleep infinity"
```

The above command uses *nvidia-docker* which is like the *docker* command except that it allows the container to directly access the GPUs on the host. The final line starts the SSH daemon and waits forever. At this point, the container can be accessed by MPI via the SSH daemon listening on port 2222.

Choose any one of the DGX-2 systems as the primary node and enter the container by running the following command. This will give you a bash prompt within the container.

```
docker exec -it tf bash
```

Confirm that this container can connect to all other containers via password-less SSH on port 2222.

```
ssh dgx2-1 hostname  
ssh dgx2-2 hostname
```

Next, test that MPI can launch processes across all DGX-2 systems.

```
mpirun --allow-run-as-root -np 2 -H dgx2-1 -H dgx2-2 hostname
```

To stop the containers and all processes within them, run the following command on each DGX-2 system

```
docker stop tf
```

Note that the script *start_containers.sh* automates some of these steps.

Appendix – Monitoring Isilon performance

INSIGHTIQ

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.

For non-production and unsupported usage, InsightIQ can be installed in Docker using instructions at <https://github.com/j-sims/docker-insightiq>.

ISILON STATISTICS CLI

For a quick way to investigate the performance of an Isilon cluster when InsightIQ is not available, there is a wealth of statistics that are available through the Isilon CLI, which can be accessed using SSH to any Isilon node.

This first command shows the highest level of statistics. Note that units are in bytes/sec so in the example below Node 1 is sending (NetOut) 2.9 GB/sec to clients.

```
isilon-1# isi statistics system --nodes all --format top  
Node   CPU   SMB FTP HTTP   NFS HDFS  Total   NetIn NetOut DiskIn DiskOut  
All 72.7% 0.0 0.0 0.0 10.0G 0.0 10.0G 304.4M 10.4G 315.4M 10.8G  
1 79.2% 0.0 0.0 0.0 2.9G 0.0 2.9G 295.0M 2.9G 70.5M 2.3G
```

2	80.6%	0.0	0.0	0.0	2.7G	0.0	2.7G	3.2M	2.7G	95.5M	2.8G
3	71.9%	0.0	0.0	0.0	2.4G	0.0	2.4G	3.4M	2.6G	75.5M	2.8G
4	59.2%	0.0	0.0	0.0	2.0G	0.0	2.0G	2.9M	2.1G	73.9M	2.9G

The following command shows more details related to NFS. All statistics are aggregated over all nodes.

```
isilon-1 # isi statistics pstat --format top
NFS3 Operations Per Second
access          2.33/s  commit          0.00/s  create          0.75/s
fsinfo          0.00/s  getattr         2.09/s  link            0.00/s
lookup          0.99/s  mkdir           0.00/s  mknod           0.00/s
noop            0.00/s  null            0.00/s  pathconf        0.00/s
read            18865.24/s  readdir         0.00/s  readdirplus     0.00/s
readlink        0.00/s  remove          0.00/s  rename          0.75/s
rmdir           0.00/s  setattr         0.00/s  statfs          0.00/s
symlink         0.00/s  write           0.75/s
Total           18872.91/s

CPU Utilization
user            1.4%
system          72.5%
idle            26.1%

OneFS Stats
In              73.81 kB/s
Out             8.96 GB/s
Total           8.96 GB/s

Network Input      Network Output      Disk I/O
MB/s              MB/s              Disk  272334.03 iops
Pkt/s            150368.20         Pkt/s  6787182.27
Errors/s          0.00              Errors/s  0.00
Read             11.73 GB/s
Write            99.63 MB/s
```

Appendix – Isilon performance testing with iPerf and FIO

iPerf is a tool for active measurements of the maximum achievable bandwidth on IP networks. It can be used to validate the throughput of the IP network path from an Isilon node to a compute node NIC. It can easily be scripted to run concurrently to allow all nodes to send or receive traffic.

FIO is a disk benchmark tool for Linux. It can be used to easily and quickly produce a storage workload that is nearly identical to the TensorFlow benchmark used in this document.

This section shows how to use iPerf and FIO.

To begin, on any DGX-2 system, create a file named `hosts` containing one host name or IP address per client. For example:

```
server1.example.com
server2.example.com
```

IPERF

Using iPerf to test Isilon to DGX-2 system performance

1. Install iPerf on all compute nodes. Note that iPerf is already installed on all Isilon nodes. All versions should match.

```
cat hosts | xargs -i -P 0 ssh root@{} yum -y install \
iperf-2.0.4-1.el7.rf.x86_64.rpm
```

2. Start iPerf servers.

```
cat hosts | xargs -i -P 0 ssh root@{} pkill -9 iperf
cat hosts | xargs -i ssh root@{} "iperf --server --daemon \
/dev/null 2>&1 &"
```

3. Start iPerf clients.

```
client_opts="-t 300 --len 65536 --parallel 2"
ssh root@isilon-1.example.com iperf -c server1.example.com ${client_opts} &
```

```
ssh root@isilon-2.example.com iperf -c server2.example.com ${client_opts} &
wait
```

4. To view the aggregate bandwidth, run the following on any Isilon node.

```
isi statistics system --format top --nodes all
```

Using iPerf to test DGX-2 system storage network performance

1. On one DGX-2 system, start multiple instances of the iPerf server.

```
seq 5001 5016 | xargs -P 0 -i iperf --server -p {}
```

2. On another DGX-2 system, start multiple instances of the iPerf client. In the command below, four instances will use the first storage interface and the next four instances will use the second storage interface.

```
seq 5001 5004 | xargs -P 0 -i iperf -t 900 -c 10.200.10.13 -p {} &
seq 5009 5012 | xargs -P 0 -i iperf -t 900 -c 10.200.22.13 -p {} &
```

FIO

The procedure below shows how to use FIO to benchmark NFS I/O from multiple clients concurrently.

1. Install FIO servers.

```
cat hosts | xargs -i -P 0 ssh root@{} sudo apt-get install fio
cat hosts | xargs -i ssh root@{} fio -version
fio-3.1
```

2. Start FIO servers.

```
cat hosts | xargs -i ssh root@{} pkill fio
cat hosts | xargs -i ssh root@{} fio --server --daemonize=/tmp/fio.pid
```

3. Create a FIO job file shown below, named `fio1.job`. Set `numjobs` to the number of GPUs per host. This job file performs I/O that is like the TensorFlow CNN benchmark. It creates 30 files per GPU and then reads them sequentially concurrently.

```
[global]
name=job1
directory=/mnt/isilon1/tmp/fio:/mnt/isilon2/tmp/fio:/mnt/isilon3/tmp/fio:/mnt/isilon4/tmp/fio:/mnt/isilon5/tmp/fio:/mnt/isilon6/tmp/fio:/mnt/isilon7/tmp/fio:/mnt/isilon8/tmp/fio:/mnt/isilon9/tmp/fio:/mnt/isilon10/tmp/fio:/mnt/isilon11/tmp/fio:/mnt/isilon12/tmp/fio:/mnt/isilon13/tmp/fio:/mnt/isilon14/tmp/fio:/mnt/isilon15/tmp/fio:/mnt/isilon16/tmp/fio
time_based=0
ramp_time=30
ioengine=libaio
numjobs=16
create_serialize=0
iodepth=32
kb_base=1000

[job1]
rw=read
nrfiles=30
size=458GB
bs=1024KiB
direct=1
sync=0
```

```
rate_iops=576
```

4. Run the FIO job.

```
mkdir -p /mnt/isilon/tmp/fio
fio --client=hosts fio1.job
```

Appendix – Isilon performance testing with TensorFlow

Note: This section uses several files that are available at https://github.com/claudiofahey/ai-benchmark-util/tree/master/storage_benchmark_tensorflow.

1. Start the TensorFlow containers as shown in Start TensorFlow containers.

2. Open a Bash terminal in any TensorFlow container.

```
docker exec -it tf bash
```

3. Edit the configuration file `storage_benchmark_tensorflow/storage_benchmark_tensorflow.yaml` as needed.

4. Start the benchmark.

```
cd storage_benchmark_tensorflow
./storage_benchmark_tensorflow.sh
```

Appendix – Switch configuration

This section shows the configuration for the switches.

Switch 1 Configuration	Switch 2 Configuration
<pre>! Version 10.5.0.0 ip vrf default ip name-server 8.8.8.8 interface breakout 1/1/1 map 100g-1x interface breakout 1/1/2 map 100g-1x interface breakout 1/1/3 map 100g-1x interface breakout 1/1/4 map 100g-1x interface breakout 1/1/5 map 100g-1x interface breakout 1/1/6 map 100g-1x interface breakout 1/1/7 map 100g-1x interface breakout 1/1/8 map 100g-1x interface breakout 1/1/9 map 40g-1x interface breakout 1/1/10 map 40g-1x interface breakout 1/1/11 map 40g-1x interface breakout 1/1/12 map 40g-1x interface breakout 1/1/13 map 40g-1x interface breakout 1/1/14 map 40g-1x interface breakout 1/1/15 map 40g-1x interface breakout 1/1/16 map 40g-1x interface breakout 1/1/17 map 100g-1x interface breakout 1/1/18 map 100g-1x interface breakout 1/1/19 map 100g-1x interface breakout 1/1/20 map 100g-1x interface breakout 1/1/21 map 100g-1x interface breakout 1/1/22 map 100g-1x interface breakout 1/1/23 map 100g-1x interface breakout 1/1/24 map 100g-1x interface breakout 1/1/25 map 100g-1x interface breakout 1/1/26 map 100g-1x interface breakout 1/1/27 map 100g-1x</pre>	<pre>! Version 10.5.0.0 ip vrf default ip name-server 8.8.8.8 interface breakout 1/1/1 map 100g-1x interface breakout 1/1/2 map 100g-1x interface breakout 1/1/3 map 100g-1x interface breakout 1/1/4 map 100g-1x interface breakout 1/1/5 map 100g-1x interface breakout 1/1/6 map 100g-1x interface breakout 1/1/7 map 100g-1x interface breakout 1/1/8 map 100g-1x interface breakout 1/1/9 map 40g-1x interface breakout 1/1/10 map 40g-1x interface breakout 1/1/11 map 40g-1x interface breakout 1/1/12 map 40g-1x interface breakout 1/1/13 map 40g-1x interface breakout 1/1/14 map 40g-1x interface breakout 1/1/15 map 40g-1x interface breakout 1/1/16 map 40g-1x interface breakout 1/1/17 map 100g-1x interface breakout 1/1/18 map 100g-1x interface breakout 1/1/19 map 100g-1x interface breakout 1/1/20 map 100g-1x interface breakout 1/1/21 map 100g-1x interface breakout 1/1/22 map 100g-1x interface breakout 1/1/23 map 100g-1x interface breakout 1/1/24 map 100g-1x interface breakout 1/1/25 map 100g-1x interface breakout 1/1/26 map 100g-1x interface breakout 1/1/27 map 100g-1x</pre>

```

interface breakout 1/1/28 map 100g-1x
interface breakout 1/1/29 map 100g-1x
interface breakout 1/1/30 map 100g-1x
interface breakout 1/1/31 map 100g-1x
interface breakout 1/1/32 map 100g-1x
hash-algorithm lag random
!
interface vlan1
  no shutdown
!
interface vlan55
  description roce
  no shutdown
!
interface port-channel1
  no shutdown
  switchport access vlan 1
  mtu 9216
!
interface mgmt1/1/1
  no shutdown
  no ip address dhcp
  ip address 172.28.10.135/26
  ipv6 address autoconfig
!
interface ethernet1/1/1
  description dgx2-1-c1
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/2
  description dgx2-1-c5
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/3
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/4
  description dgx2-2-c1
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/5
  description dgx2-2-c5
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/6

```

```

interface breakout 1/1/28 map 100g-1x
interface breakout 1/1/29 map 100g-1x
interface breakout 1/1/30 map 100g-1x
interface breakout 1/1/31 map 100g-1x
interface breakout 1/1/32 map 100g-1x
hash-algorithm lag random
!
interface vlan1
  no shutdown
!
interface vlan55
  description roce
  no shutdown
!
interface port-channel1
  no shutdown
  switchport access vlan 1
  mtu 9216
!
interface mgmt1/1/1
  no shutdown
  no ip address dhcp
  ip address 172.28.10.136/26
  ipv6 address autoconfig
!
interface ethernet1/1/1
  description dgx2-1-c2
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/2
  description dgx2-1-c6
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/3
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/4
  description dgx2-2-c2
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/5
  description dgx2-2-c6
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/6

```

```

no shutdown
switchport access vlan 55
mtu 9216
flowcontrol receive off
!
interface ethernet1/1/7
description dgx2-3-c1
no shutdown
switchport access vlan 55
mtu 9216
flowcontrol receive off
!
interface ethernet1/1/8
description dgx2-3-c5
no shutdown
switchport access vlan 55
mtu 9216
flowcontrol receive off
!
interface ethernet1/1/9:1
description isilon-4-1
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/10:1
description isilon-3-1
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/11:1
description isilon-2-2
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/12:1
description isilon-1-2
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/13:1
description isilon-8-1
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/14:1
description isilon-7-1
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/15:1
description isilon-6-2
no shutdown
switchport access vlan 1
mtu 9216

```

```

no shutdown
switchport access vlan 55
mtu 9216
flowcontrol receive off
!
interface ethernet1/1/7
description dgx2-3-c2
no shutdown
switchport access vlan 55
mtu 9216
flowcontrol receive off
!
interface ethernet1/1/8
description dgx2-3-c6
no shutdown
switchport access vlan 55
mtu 9216
flowcontrol receive off
!
interface ethernet1/1/9:1
description isilon-4-2
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/10:1
description isilon-3-2
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/11:1
description isilon-2-1
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/12:1
description isilon-1-1
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/13:1
description isilon-8-2
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/14:1
description isilon-7-2
no shutdown
switchport access vlan 1
mtu 9216
!
interface ethernet1/1/15:1
description isilon-6-1
no shutdown
switchport access vlan 1
mtu 9216

```

```

!
interface ethernet1/1/16:1
  description isilon-5-2
  no shutdown
  switchport access vlan 1
  mtu 9216
!
interface ethernet1/1/17
  description isl-1-1
  no shutdown
  channel-group 1
  no switchport
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/18
  description isl-1-2
  no shutdown
  channel-group 1
  no switchport
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/19
  description mgmt-1-2
  no shutdown
  no switchport
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/20
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/21
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/22
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/23
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/24
  description dgx2-3-s1
  shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off

```

```

!
interface ethernet1/1/16:1
  description isilon-5-1
  no shutdown
  switchport access vlan 1
  mtu 9216
!
interface ethernet1/1/17
  description isl-2-1
  no shutdown
  channel-group 1
  no switchport
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/18
  description isl-2-2
  no shutdown
  channel-group 1
  no switchport
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/19
  description mgmt-2-2
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/20
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/21
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/22
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/23
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/24
  description dgx2-3-s2
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off

```



```

!
interface ethernet1/1/25
  description dgx2-3-c7
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/26
  description dgx2-3-c3
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/27
  description dgx2-2-s1
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/28
  description dgx2-2-c7
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/29
  description dgx2-2-c3
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/30
  description dgx2-1-s1
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/31
  description dgx2-1-c7
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/32
  description dgx2-1-c3
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/33
  no shutdown
  switchport access vlan 1

```

```

!
interface ethernet1/1/25
  description dgx2-3-c8
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/26
  description dgx2-3-c4
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/27
  description dgx2-2-s2
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/28
  description dgx2-2-c8
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/29
  description dgx2-2-c4
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/30
  description dgx2-1-s2
  no shutdown
  switchport access vlan 1
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/31
  description dgx2-1-c8
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/32
  description dgx2-1-c4
  no shutdown
  switchport access vlan 55
  mtu 9216
  flowcontrol receive off
!
interface ethernet1/1/33
  no shutdown
  switchport access vlan 1

```

<pre> flowcontrol receive off ! interface ethernet1/1/34 no shutdown switchport access vlan 1 mtu 9216 flowcontrol receive off ! management route 0.0.0.0/0 172.28.10.130 ! snmp-server community public ro ! ntp server pool.ntp.org </pre>	<pre> flowcontrol receive off ! interface ethernet1/1/34 no shutdown switchport access vlan 1 mtu 9216 flowcontrol receive off ! management route 0.0.0.0/0 172.28.10.130 ! snmp-server community public ro ! ntp server pool.ntp.org </pre>
--	--

Appendix – Collecting system metrics with Prometheus and Grafana

While performing benchmarks on a distributed system such as this, it is valuable to collect and display all relevant metrics such as CPU utilization, GPU utilization, memory usage, network I/O and more. This can be performed with a variety of tools. For this document, data was collected using Prometheus and Grafana. Although these are easy to install in Docker, the instructions below show how to install these on a single-node Kubernetes system deployed with Minikube.

Note: This section uses several files that are available at <https://github.com/claudiofahey/ai-benchmark-util/tree/dgx2>.

1. Install Kubernetes

- a. Install Minikube on an administrative server (not the DGX-2 system). You can use the steps below to start a Minikube cluster directly on Linux. For more information, refer to <https://kubernetes.io/docs/tasks/tools/install-minikube/>.

```
minikube start --vm-driver=none --kubernetes-version='v1.14.3'
```

- b. It is often convenient to start the Kubernetes Dashboard as follows.

```
minikube dashboard --url
kubectl edit svc -n kube-system kubernetes-dashboard
```

- c. Change this service to use a NodePort. This will allow remote access to the Kubernetes Dashboard.

```
kubectl get svc -n kube-system kubernetes-dashboard -o yaml
```

- d. Open your browser to the address *http://admin-host:NodePort*.

2. Install Helm

- a. Follow the steps at <https://helm.sh/docs/intro/install/>.

3. Install Prometheus

- a. Change the current directory.

```
cd ai-benchmark-util/prometheus
```

- b. Edit `values.yaml` with IP addresses of DGX-2 systems and Ethernet switches to monitor.

- c. Install Prometheus using Helm.

```
./install.sh
```

- d. Install Prometheus Node Exporter and NVIDIA GPU Exporter on DGX-2 systems

- e. Change the current directory.

```
cd ai-benchmark-util/ansible
```

- f. Edit `inventory.yaml` with the IP addresses or hostnames of DGX-2 systems.

- g. Run the following commands.

```
sudo apt install ansible
ansible-galaxy install cloudalchemy.node-exporter
ansible-playbook --user root --inventory inventory.yaml \
playbook_monitoring.yaml -v
```

4. Install Prometheus SNMP Exporter.

- a. Run the following commands.

```
cd ai-benchmark-util/prometheus-snmp-exporter
./install.sh
```

5. Install Grafana

- a. Run the following commands.

```
helm upgrade --install grafana stable/grafana --set service.type=NodePort
```

- b. Get the generated Grafana password using the command below.

```
kubectl get secret --namespace default grafana \
-o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

- c. Determine the NodePort number.

```
kubectl get svc -n default grafana -o yaml
```

- d. Open your browser to the address *http://admin-host:NodePort*.

- e. Login to Grafana with username admin and the password from the above command.

- f. Add a data source:

- i. Type: Prometheus

- ii. HTTP URL: *http://prometheus-server.default.svc.cluster.local:80*

- g. Import the Grafana dashboards from *ai-benchmark-util/grafana*.

Figure 20 shows a sample Grafana dashboard.

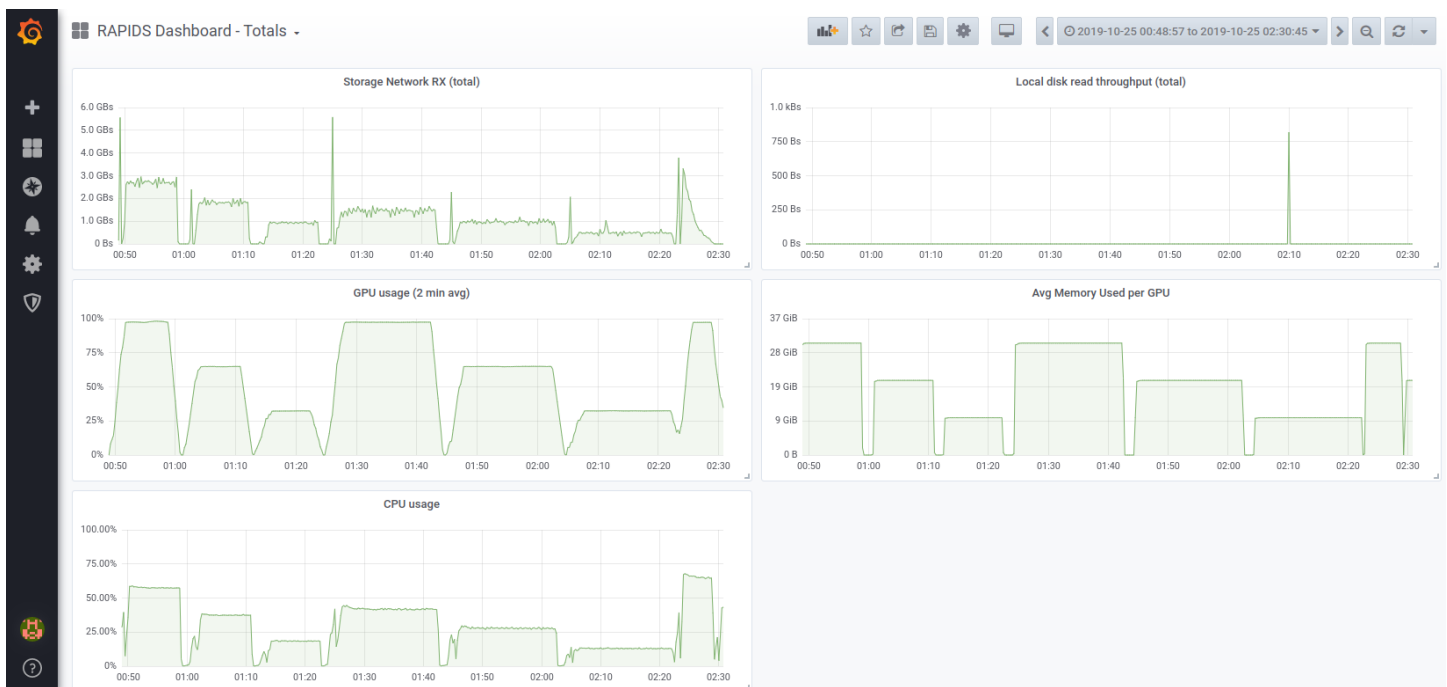


Figure 20: Grafana Dashboard

References

Name	Link
AI Benchmark Utilities	https://github.com/claudiofahey/ai-benchmark-util/tree/dgx2
Dell EMC Isilon F800	https://www.dellemc.com/en-be/collaterals/unauth/data-sheets/products/storage/h15963-ss-isilon-all-flash.pdf
Dell EMC Isilon OneFS Best Practices	https://www.emc.com/collateral/white-papers/h16857-wp-onefs-best-practices.pdf
Dell EMC Isilon OneFS SmartFlash	https://www.dellemc.com/resources/en-us/asset/white-papers/products/storage/h13249-isilon-onefs-smartflash-wp.pdf
Dell EMC Isilon OneFS Technical Overview	https://www.dellemc.com/en-tz/collaterals/unauth/technical-guides-support-information/products/storage/h10719-isilon-onefs-technical-overview-wp.pdf
Dell EMC Isilon Storage Tiering	https://www.dellemc.com/resources/en-us/asset/white-papers/products/storage/h8321-wp-smartpools-storage-tiering.pdf
Dell EMC Isilon and NVIDIA DGX-1 servers for deep learning	https://www.dellemc.com/resources/en-us/asset/white-papers/products/storage/Dell EMC Isilon and NVIDIA DGX 1 servers for deep learning.pdf
Dell EMC Networking OS10 Enterprise Edition User Guide	https://topics-cdn.dell.com/pdf/force10-s3048-on_connectivity-guide4_en-us.pdf
Dell EMC PowerSwitch S5232F-ON	https://i.dell.com/sites/csdocuments/Product_Docs/en/Dell-EMC-Networking-s5200-series-specsheet.pdf
Gartner	https://www.gartner.com/en/newsroom/press-releases/2019-08-05-gartner-says-ai-augmentation-will-create-2point9-trillion-of-business-value-in-2021
Horovod	https://github.com/horovod/horovod
ImageNet	http://www.image-net.org/challenges/LSVRC/2012/
MPI All Reduce	http://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/
NVIDIA DGX-2 system	https://www.nvidia.com/dgx2
NVIDIA GPU Cloud (NGC) TensorFlow image	https://ngc.nvidia.com/registry/nvidia-tensorflow
TensorFlow	https://github.com/tensorflow/tensorflow
TensorFlow Benchmarks	https://github.com/tensorflow/benchmarks
TensorFlow Inception	https://github.com/tensorflow/models/tree/master/research/inception