Whitepaper

Dell EMC Isilon and Dell EMC DSS 8440 Servers for Deep Learning

This document demonstrates how the Dell EMC Isilon F800 All-Flash Scale-out NAS, Dell EMC DSS 8440 servers with NVIDIA Tesla® V100 GPUs, and Dell EMC POWERSWITCH S5232 switches can be used to accelerate and scale deep learning training workloads. The results of industry-standard image classification benchmarks using TensorFlow, MXNet, and PyTorch are included.

July 2019

Table of Contents

Revisions

Date	Description	Author
July 2019	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 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 EMC 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.

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 artificial intelligence 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. But, turning the promise of AI into real value isn't as easy as flipping a switch. 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. Dell EMC and NVIDIA are at the forefront of AI providing the technology that makes tomorrow possible today. Over the last few years we 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 Graphic Processing Units (GPUs) for parallel model training, and scale-out file systems to support the concurrency, performance, and Petabyte scale of the unstructured image and video data sets.

This document presents a new DL reference architecture using Dell EMC Isilon F800 storage, Dell EMC DSS 8440 servers, and Dell EMC Networking S5232 switches. 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, MXNet, and PyTorch are included.

Deep learning dataflow

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

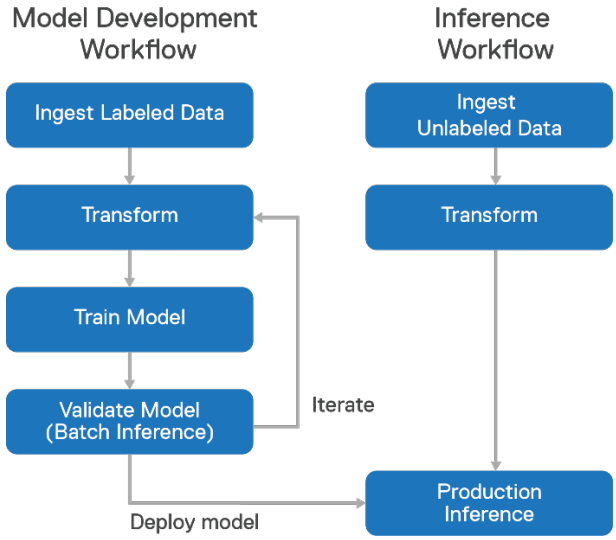


Figure 1: Common DL Workflows: Model development and inference

The workflow steps are defined and detailed below. Note that the Isilon storage and DSS 8440 server 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.

1. **Ingest Labeled Data -** In this step, 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 -** In this phase, 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 proven to work well. To provide an example, Figure 2 shows the high-level workflow 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.

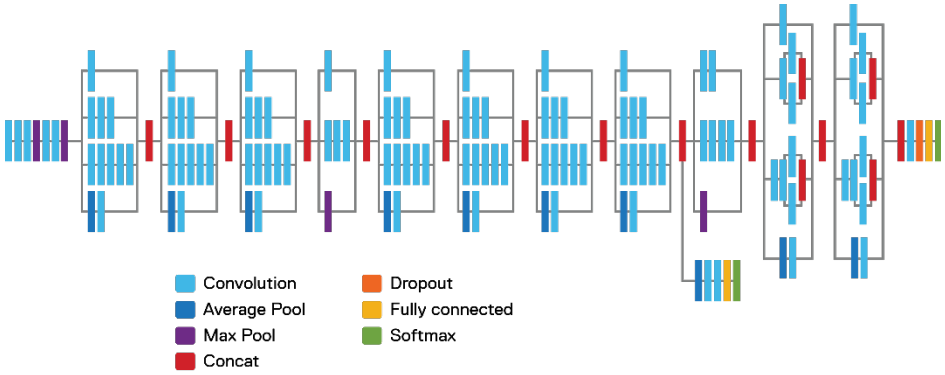


Figure 2: Inception-v3 model architecture

4. **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 ground truth label. This is often referred to as inference but keep in mind that this is a distinct step from production inference.
5. **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 make up the solution. Note that in a customer deployment, the number of DSS 8440 servers 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.

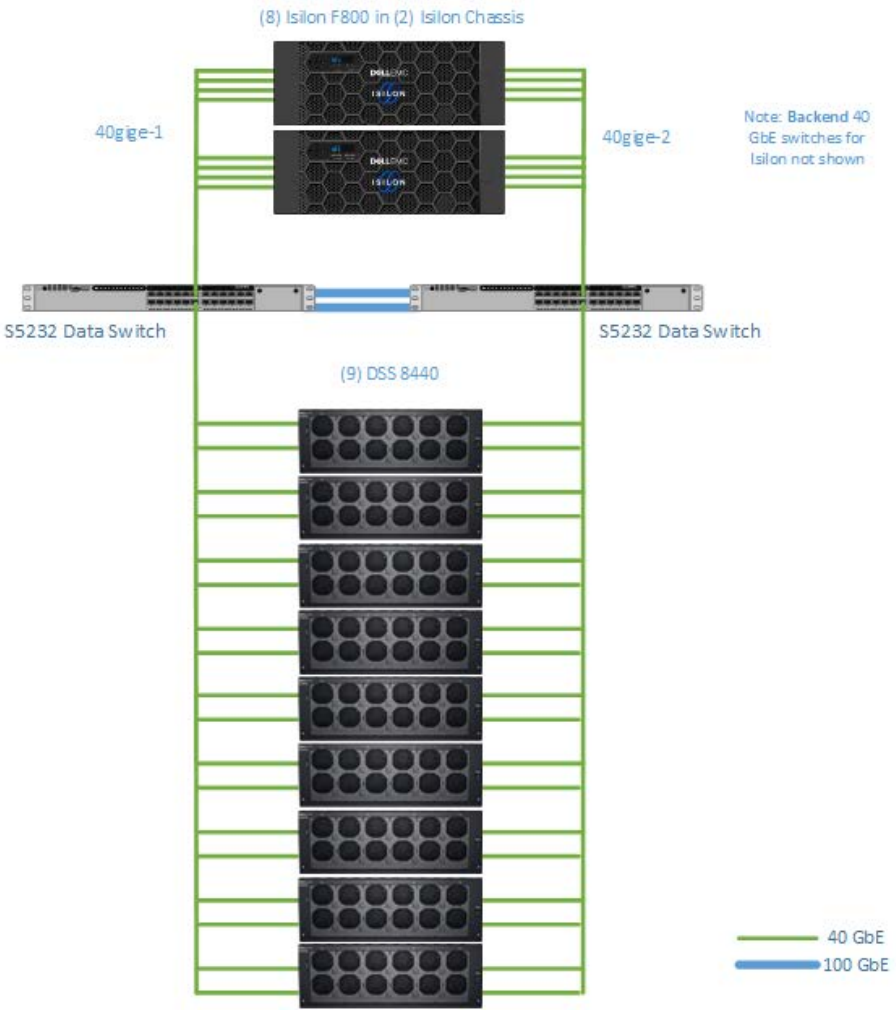


Figure 3: Reference Architecture

Figure 4 illustrates the system as it was tested and benchmarked. Only one DSS 8440 server, containing (10) NVIDIA V100 GPUs, was tested as part of this document.

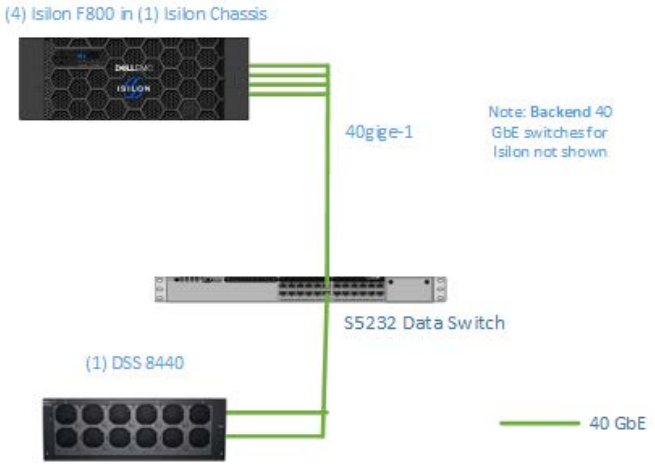


Figure 4: Solution Architecture as Tested

STORAGE: DELL EMC ISILON F800

Dell EMC Isilon F800 represents the 6th generation of hardware built to run the well-proven and massively scalable OneFS operating system. Each F800 chassis, shown in Figure 5, contains four storage nodes, 60 high performance Solid State Drives (SSDs), and eight 40 Gigabit Ethernet 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 5: Isilon F800 chassis, containing four storage nodes

In the solution tested in this document, four F800 nodes, in one chassis, were used.

Dell EMC Isilon F800 has the following features.

Low latency, high throughput and massively parallel IO 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 datasets from tens of TBs to tens of PBs on AI platforms such as TensorFlow, MXNet, PyTorch, 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 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
- 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 interface, 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 segregating 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 a number of 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 (or colder) 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 much lower latency than accessing hard disk drives.

The OneFS caching subsystem is coherent across the cluster. This means that if the same content exists in the private caches of multiple nodes, this cached data is consistent across all instances.

OneFS uses up to three levels of read cache, plus an NVRAM-backed write cache, or coalescer. These, and their high-level interaction, are illustrated in Figure 6.

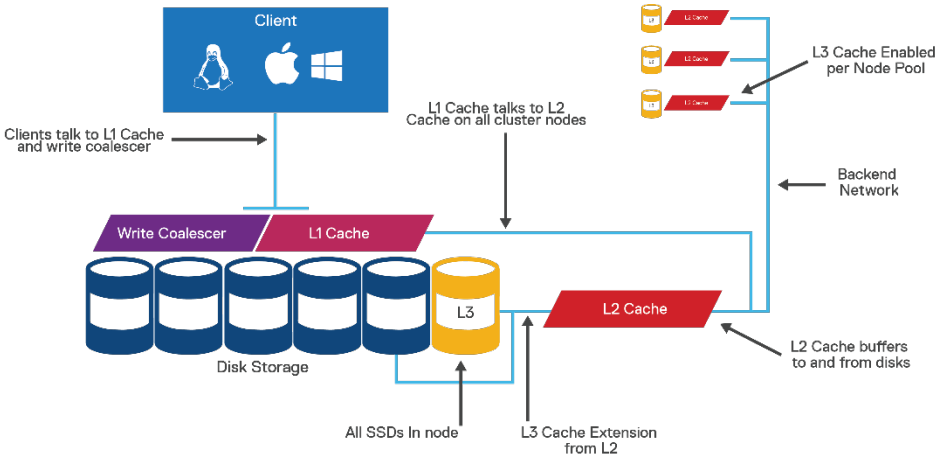


Figure 6: OneFS Caching Architecture

File Reads

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. SmartRead can create a data pipeline from L2 cache, prefetching into a local L1 cache on the captain node. 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.

Intelligent caching provided by SmartRead allows for very high read performance with high levels of concurrent access. Importantly, it is faster for Node 1 to get file data from the cache of Node 2 (over the low-latency cluster interconnect) than to access its own local disk. The SmartRead algorithms control how aggressive the pre-fetching is (disabling pre-fetch for random-access cases), how long data stays in the cache, and optimizes where data is cached. This optimized file read logic is visualized below in Figure 7.

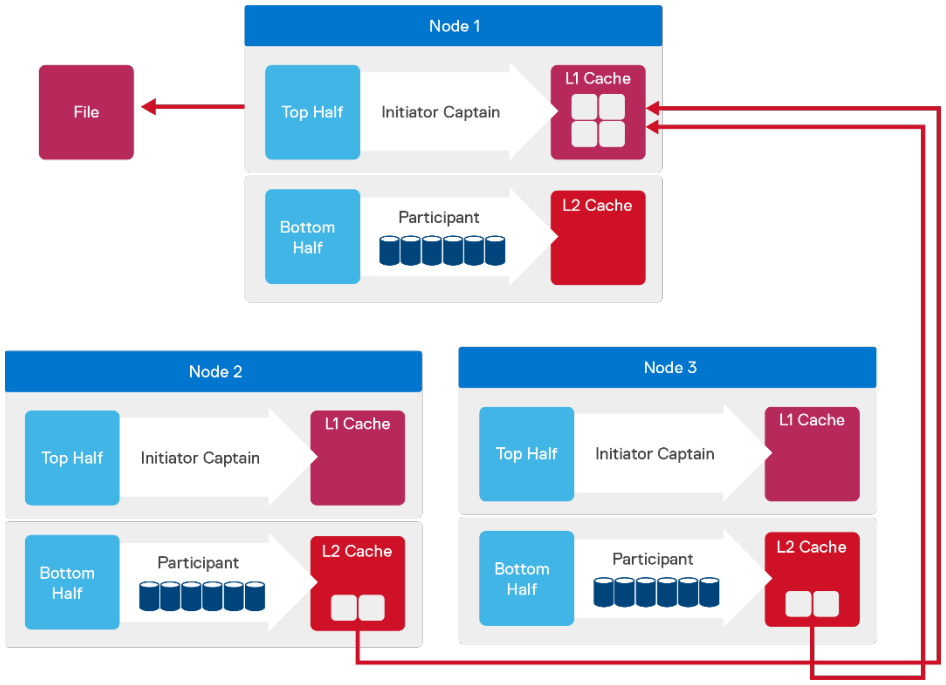


Figure 7: A File Read Operation on a 3-node Isilon Cluster

Locks and concurrency

Efficient locking is critical to support the efficient parallel IO profile demanded by many iterative DL workloads enabling concurrent file read access up into the millions. OneFS has a fully distributed lock manager that coordinates locks on data across all nodes in a storage cluster. The locking 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.

For more detailed technical overview of Isilon OneFS, refer to

https://www.emc.com/collateral/hardware/white-papers/h10719-isilon-onefs-technical-overview-wp.pdf.

COMPUTE: DELL EMC DSS 8440 SERVER

The DSS 8440 is a 4U 2-socket accelerator-optimized **s**erver designed to deliver exceptionally high compute performance. Its open architecture maximizes customer choice for machine learning infrastructure while also delivering best of breed technology (#1 server provider & the #1 GPU provider). It lets you tailor your machine learning infrastructure to your specific needs – without lock-in.

With a choice of 4, 8 or 10 of the industry-leading NVIDIA® Tesla® V100 Tensor Core GPUs, combined with 2 Intel CPUs for system functions, a high performance switched Gen 4 PCIe fabric for rapid IO and up to 10 local NVMe and SATA drives for optimized access to data, this server has both the performance and flexibility to be an ideal solution for machine learning training – as well as other compute-intensive workloads like simulation, modeling and predictive analysis in engineering and scientific environments.

Figure 8: Dell EMC DSS 8440 server with 10 Tesla V100 GPUs

When the DSS 8440 is configured with NVIDIA V100 GPUs you get the best of both worlds – working with the world's #1 server provider (Dell EMC) and the industry's #1 provider of GPU accelerators (NVIDIA). In addition, you can take advantage of the work NVIDIA has done with NVIDIA GPU Cloud (NGC), a program that offers a registry for pre-validated, pre-optimized containers for a wide range of machine learning frameworks, including TensorFlow, PyTorch, and MXNet. Along with the performance-tuned NVIDIA AI stack these pre-integrated containers include NVIDIA® CUDA® Toolkit, NVIDIA deep learning libraries, and the top AI software. They help data scientists and researchers rapidly build, train, and deploy AI models to meet continually evolving demands

NETWORKING: DELL EMC POWERSWITCH S5232F-ON SWITCH

The S5200-ON series 25/100GbE fixed switches comprise Dell EMC's latest disaggregated hardware and software data center networking solutions, providing state-of-the-art, high-density 25/100GbE 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.

The S5232 model used in this reference architecture has 32 QSFP28 ports, each of which supports direct 100 GbE and 10/25/40/50GbE in breakout mode.

Figure 9: Dell EMC POWERSWITCH S5232F-ON switch: 32 x 40/100GbE QSFP28 ports

BILL OF MATERIALS

Table 1 shows the bill of materials of the system as tested and benchmarked.

Component	Purpose	Quantity
Dell EMC Isilon F800 Chassis 96 TB SSD 1 TB RAM 4x1GE, 8x40GE Networking	Storage	1 4U Chassis (4 nodes)
Dell EMC DSS 8440 server (10) Tesla V100-PCIE-32GB GPUs (2) Intel Xeon Gold 6148 CPU @ 2.40 GHz 768 GB RAM	Compute server with 10 GPUs	1
Dell EMC Networking S5232	40/100 Gb Ethernet Data switch	1

Table 1: Bill of Materials

SOFTWARE VERSIONS

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

Component	Version
Isilon - OneFS	8.1.3.0
NVIDIA GPU Cloud Image	nvcr.io/nvidia/tensorflow:19.03-py3 and 18.09-py3
TensorFlow Benchmarks	https://github.com/tensorflow/benchmarks/commit/4828965 and https://github.com/claudiofahey/benchmarks/commit/85f0b6b
TensorFlow Benchmark Util	https://github.com/claudiofahey/tensorflow-benchmark-util/commit/73e7f41
DSS 8440 – Ubuntu	16.04.5 LTS
DSS 8440 – NVIDIA Driver	418.74
DSS 8440 – CUDA	10.1
S5232 OS	10.4.3E

Table 2: Software Versions

Deep learning training performance and analysis

BENCHMARK METHODOLOGY

To evaluate the solution's performance with a variety of workloads, multiple benchmark frameworks were used including MXNet, PyTorch, and two different TensorFlow benchmarks.

TensorFlow CNN Benchmarks

Various benchmarks from the TensorFlow Benchmarks repository were carefully executed (see https://www.tensorflow.org/performance/benchmarks#methodology). 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 GB[1]. 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 B). 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.

The TensorFlow CNN Benchmarks framework was used to train the models GoogleNet, Inception V3, ResNet-50 (V1.0), ResNet-152, and VGG16. However, it was not used to train ResNet-50 V1.5.

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 DSS 8440 server. Of course, this is not surprising since each DSS 8440 server has 768 GB 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 DSS 8440 server RAM, but larger

[1] All unit prefixes use the SI standard (base 10) where 1 GB is 1 billion bytes.

than the 1 TB of coherent shared cache available across the 4-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. 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.

In Table 3, rows with a dataset of "ImageNet 2012 150x" indicate that the 22.2 TB dataset was used.

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 3.5% cache hit rate from the Linux cache on the DSS 8440 server and a 4.5% cache hit rate from Isilon. Conversely, we can say that 95.5% of the dataset is read from Isilon's SSDs. If the dataset were twice as large (44 TB), 97.8% of the dataset would need to be read from Isilon's SSDs, which is only a slightly faster rate than what we have measured with the 22 TB dataset.

There are a variety of ways to parallelize model training to take advantage of multiple GPUs across multiple servers. In our tests, we used MPI and Horovod.

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 DSS 8440 servers 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 (V1.0) training with 10 GPUs.

```
mpirun \
--n 10 \
-allow-run-as-root \
--host localhost:10 \
--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 ob1 \
-mca btl ^openib \
-mca btl_tcp_if_include enp132s0 \
-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 \
./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=500 \
--nodistortions \
--num_gpus=1 \
--device=gpu \
--force_gpu_compatible=True \
--data_format=NCHW \
--use_fp16=True \
--use_tf_layers=False \
--data_name=imagenet \
--use_datasets=True \
--num_intra_threads=1 \
--num_inter_threads=10 \
--datasets_prefetch_buffer_size=20 \
--datasets_num_private_threads=2 \
--train_dir=/imagenet-scratch/train_dir/2019-05-17-18-15-15-resnet50\
--sync_on_finish=True \
--summary_verbosity=1 \
--save_summaries_steps=100 \
--save_model_secs=600 \
--variable_update=horovod \
--horovod_device=gpu
```

The script ***round_robin_mpi.py*** was used to add the ***--data_dir*** parameter that distributed each process across four different mount points and thus to different Isilon nodes. Note that when testing the Linux Cache performance, only a single mount point was used.

For the other models, only the *--**model*** parameter was changed. For different numbers of GPUs, only the *--**np*** parameter was changed. Note that the ***-map-by*** slot setting causes MPI to use all 10 GPUs (slots) on a DSS 8440 server before it begins using the next DSS 8440 server.

TensorFlow ResNet-50 V1.5

A different TensorFlow application was used to benchmark ResNet-50 V1.5 training. Compared to ResNet-50 V1.0, this newer version of ResNet-50 is 0.5% more accurate but 5% slower (images per second). It uses the same TFRecord files as the TensorFlow CNN Benchmarks.

The benchmark was executed using the following commands:

```
root@dss8440-1:/mnt/isilon/data#
git clone https://github.com/NVIDIA/DeepLearningExamples
cd DeepLearningExamples/TensorFlow/Classification/RN50v1.5

bash scripts/docker/build.sh

nvidia-docker run --rm -it --ipc=host \
--shm-size=1g \
--ulimit memlock=-1 \
--ulimit stack=67108864 \
-v $PWD:/workspace/rn50v15_tf/ \
-v /mnt:/mnt \
-v /data:/data \
rn50v15_tf \
bash

root@8a87621b1cec:/workspace/rn50v15_tf#
mpiexec \
--allow-run-as-root \
--bind-to socket \
-np 10 \
python \
./main.py \
--num_iter=90 \
--iter_unit=epoch \
```

```
--data_dir=/mnt/isilon1/data/imagenet-scratch/tfrecords \
--batch_size=256 \
--use_tf_amp \
--results_dir=/tmp/results
```

See the ResNet-50 V1.5 for TensorFlow Quick Start Guide additional details.

MXNet ResNet-50 V1.5

MXNet is a different DL framework and it has been highly optimized to execute ResNet-50 V1.5 on the NVIDIA V100 GPU. MXNet requires a different procedure to prepare the input data files.

The benchmark was executed using the following commands:

```
root@dss8440-1:/mnt/isilon/data#
git clone https://github.com/NVIDIA/DeepLearningExamples
cd DeepLearningExamples/MxNet/Classification/RN50v1.5

ls -lh /mnt/isilon/data/mxnet_mlperf/mxnet_recordio
total 154G
-rw-r--r-- 1 root root  24M May 10 12:53 train.idx
-rw-r--r-- 1 root root 111G May 10 12:53 train.rec
-rw-r--r-- 1 root root 809K May 10 12:53 val.idx
-rw-r--r-- 1 root root 4.8G May 10 12:34 val.rec

nvidia-docker run --rm -it --ipc=host \
-v $PWD:/workspace/resnet50 \
-v /mnt/isilon1/data/mxnet_mlperf/mxnet_recordio:\
/data/imagenet/train-val-recordio-passthrough \
nvcr.io/nvidia/mxnet:19.03-py3

root@ba3cbebeb640:/workspace#
cd resnet50
./runner -n 8 -b 256 --model-prefix model
```

See the ResNet-50 V1.5 for MXNet Quick Start Guide additional details.

PyTorch ResNet-50 V1.5

PyTorch is yet another DL framework. Unlike TensorFlow and MXNet, it reads the input data directly from individual JPEG files. It does not require any preprocessing steps to prepare the input data files.

The benchmark was executed using the following commands:

```
root@dss8440-1:/mnt/isilon/data#
git clone https://github.com/NVIDIA/DeepLearningExamples
cd DeepLearningExamples/PyTorch/Classification/RN50v1.5

nvidia-docker run --rm -it --ipc=host \
-v $PWD:/workspace/resnet50 \
-v /mnt:/mnt \
-v /data:/data \
nvcr.io/nvidia/pytorch:19.03-py3

root@d9887c15a705:/workspace#
cd resnet50
python \
./main.py \
--arch resnet50 \
-c fanin \
--label-smoothing 0.1 \
--fp16 \
--static-loss-scale 256 \
/mnt/isilon1/data/pytorch
```

See the ResNet-50 V1.5 for PyTorch Quick Start Guide additional details.

BENCHMARK RESULTS

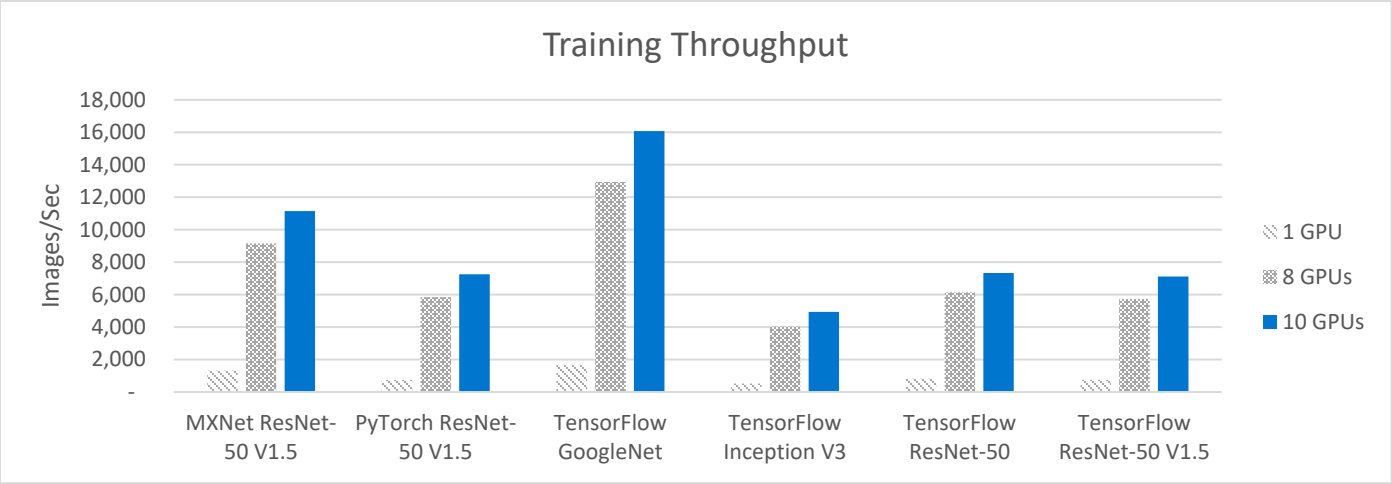


Figure 10: Model Development – Training Benchmark Results Chart
(Only results with container 19.03-py3 are shown)

Benchmark Framework	Model	Images/Sec	GPUs	Container	Batch Size	Dataset
TensorFlow CNN Benchmarks	GoogleNet	1,658	1	19.03-py3	1024	ImageNet2012
		12,937	8	19.03-py3	1024	ImageNet2012
		16,076	10	19.03-py3	1024	ImageNet2012
	Inception V3	520	1	19.03-py3	384	ImageNet2012
		4,035	8	19.03-py3	384	ImageNet2012
		4,939	10	19.03-py3	384	ImageNet2012
		3,647	8	18.09-py3	256	ImageNet2012 150x
	ResNet-50	819	1	19.03-py3	512	ImageNet2012
		6,142	8	19.03-py3	512	ImageNet2012
		7,333	10	19.03-py3	512	ImageNet2012
		801	1	18.09-py3	256	ImageNet2012 150x
		6,036	8	18.09-py3	256	ImageNet2012 150x
		7,220	10	18.09-py3	256	ImageNet2012 150x
	ResNet-152	2,611	8	18.09-py3	256	ImageNet2012 150x
	VGG16	2,895	8	18.09-py3	256	ImageNet2012 150x
TensorFlow	ResNet-50 V1.5	752	1	19.03-py3	512	ImageNet2012
		5,731	8	19.03-py3	512	ImageNet2012
		7,112	10	19.03-py3	512	ImageNet2012
MXNet	ResNet-50 V1.5	1,318	1	19.03-py3	256	ImageNet2012
		9,155	8	19.03-py3	256	ImageNet2012
		11,151	10	19.03-py3	256	ImageNet2012
PyTorch	ResNet-50 V1.5	740	1	19.03-py3	512	ImageNet2012
		5,844	8	19.03-py3	512	ImageNet2012
		7,251	10	19.03-py3	512	ImageNet2012

Table 3: Model Development – Training Benchmark Results

For all tests in Table 3, mixed precision with FP16 was used. Tests with 1 and 8 GPUs were performed with 8 Tesla V100-PCIE-32GB GPUs installed in the DSS 8440 server.

There are a few conclusions that we can make from the benchmarks represented in Table 3.

- The maximum sustained throughput that was pulled from Isilon occurred with ResNet-50 and 10 GPUs. The total storage throughput was 831 MB/sec. This is nowhere near the maximum capability of the Isilon cluster used in this test, which can exceed 15,000 MB/sec, depending on the I/O pattern.
- For all tests shown above, each GPU had 97% utilization or higher. This indicates that the GPU was the bottleneck.
- The maximum CPU utilization on the DSS 8440 server was 60%. This occurred with GoogleNet.

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. Some Isilon models also have an L3 cache. The L3 cache system will store frequently-used data on the SSDs within each Isilon node, avoiding I/O to slower HDDs. The F800 all-flash array does not have L3 cache as all data is on SSDs.

Linux Buffer Cache

A DSS 8440 server, 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 TensorFlow training pipeline, it is helpful to understand the key parts of the pipeline shown in Figure 10.

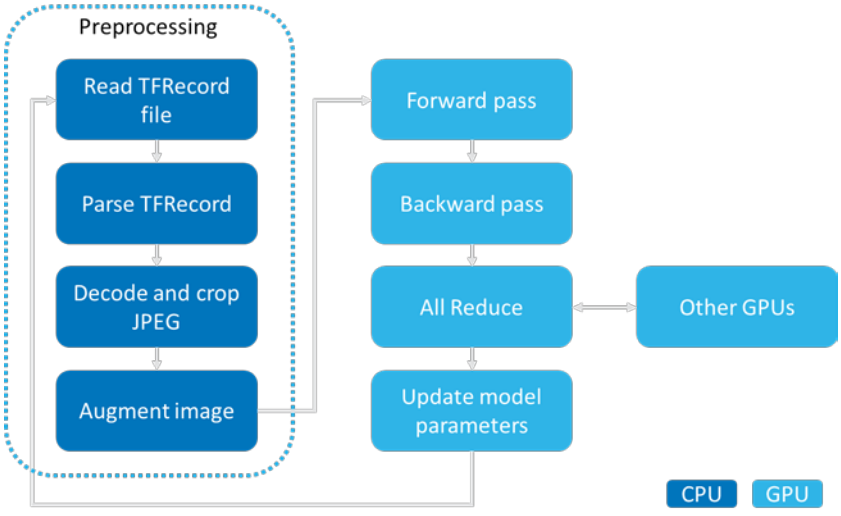


Figure 11: TensorFlow Training pipeline

1. Preprocessing

a. (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.

b. (CPU) Parse the TFRecord file into individual records and fields. The JPEG-encoded image is in one of these fields.

c. (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.

d. (CPU) The cropped image is resized to 299x299 (Inception only) or 224x224 (all other models).

e. (CPU) The image is randomly flipped horizontally or vertically.

f. (CPU) If distortions are enabled, the image brightness is randomly adjusted. Note that distortions were disabled for all benchmarks described in this document.

g. (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.

    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 DSS 8440 server.

FLOATING POINT PRECISION (FP16 VS. FP32)

The NVIDIA Tesla V100 GPU contains a new type of processing core called Tensor Core which supports mixed precision training. Although many high-performance computing (HPC) applications require high precision computation with FP32 (32-bit floating point) or FP64 (64-bit floating point), DL researchers have found they are able to achieve the same inference accuracy with FP16 (16-bit floating point) as can be had with FP32. In this document, mixed precision training which includes FP16 and FP32 representations is denoted as "FP16" training.

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

All benchmark results in this document were obtained using FP16.

Solution sizing guidance

DL workloads vary significantly with respect to the demand for compute, memory, disk and IO 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 following results in Table 4 show a few such data points:

Storage Performance Demanded	Benchmark	Required Storage Throughput per V100 GPU (MB/sec/GPU)	V100 GPUs per Isilon Node	
			F800	H500
Low	Training, Small Images, ResNet-50	90	27	7
Medium	Inference, Small Images, ResNet-50	160	20	5
High	Training, Large Images, Inception-v4	300	9	2

Table 4: Sizing consideration based on benchmarked workloads

As used in this section, "small" images are JPEGs with an average size of 115 KB and "large" images have an average size of 1.3 MB.

To illustrate, training of ResNet-50 with small images and 80 GPUs would need a storage system that can read 80 * 90 = 7200 MB/sec which can be handled by 2.9 Isilon F800 nodes. As another example, training of Inception-v4 with large images and 72 GPUs would need 72 * 300 = 21,600 MB/sec which can be handled by eight Isilon F800 nodes.

This sizing guidance is based on prior work detailed in Dell EMC Isilon and NVIDIA DGX-1 Servers for Deep Learning. It has been updated to account for the increase in training performance measured on the DSS 8440 with the most recent NGC container.

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 4, 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. 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 datasets 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.

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 discussed key features of Isilon that make it a powerful persistent storage for DL solutions. We presented a high-performance architecture for DL by combining Dell EMC DSS 8440 servers with Tesla V100 GPUs, Dell EMC Isilon F800 All-Flash NAS storage, and Dell EMC POWERSWITCH S5232 switches. This new reference architecture extends the commitment that Dell EMC has to making AI simple and accessible to every organization with our unmatched set of offerings. 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 IO to disk. We also monitored and reported the CPU, GPU utilization and memory statistics that demonstrated that the GPU resources were fully utilized while IO 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 Dell EMC DSS 8440 servers.

It is important to point out that DL algorithms have a diverse set of requirements with various compute, memory, IO, 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 set 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 10s 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 and/or to scale up to 58 PB per cluster

In summary, Isilon-based DL solutions deliver the capacity, performance, and high concurrency to eliminate the IO 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.

Appendices

Appendix A System configuration

ISILON

Configuration

The Isilon cluster configuration is simple yet it provides excellent performance. Below shows the configuration of Isilon as it was when it was tested and benchmarked for this document. On each of the four Isilon F800 nodes, only one 40 GbE port was connected. For management, a single 1 GbE port was used.

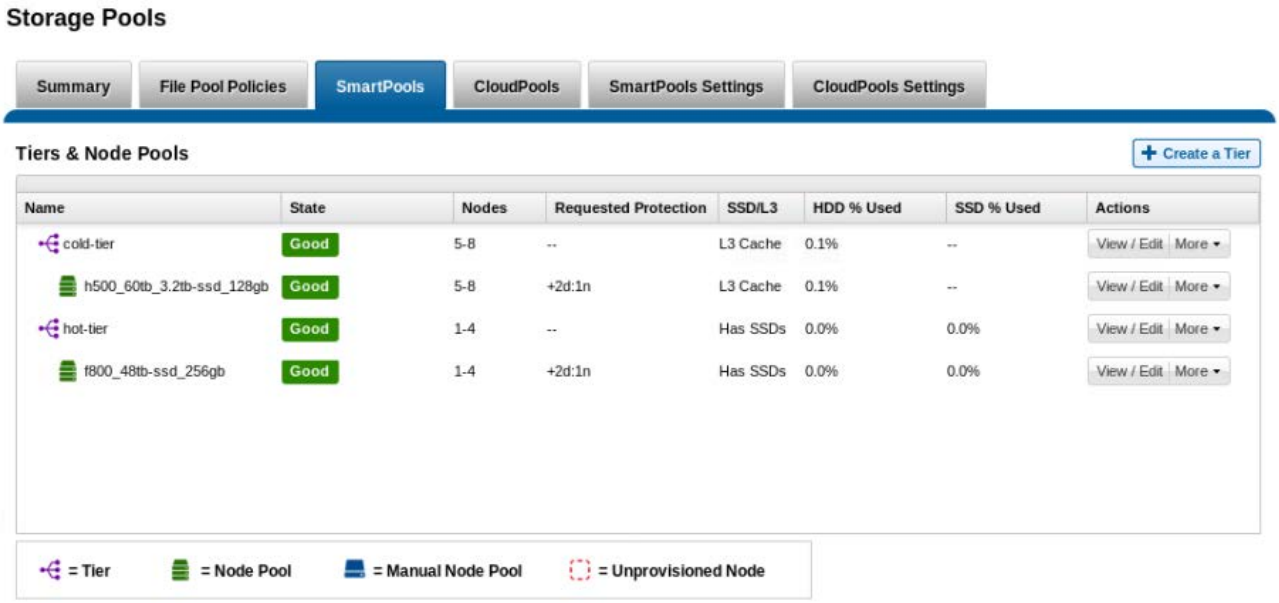
```
# isi network interfaces list
LNN  Name     Status     Owners                  IP Addresses
-------------------------------------------------------------
1    40gige-1 Up         groupnet0.subnet1.pool1 172.16.1.11
1    40gige-2 No Carrier -                       -
1    mgmt-1   No Carrier -                       -
2    40gige-1 Up         groupnet0.subnet1.pool1 172.16.1.12
2    40gige-2 No Carrier -                       -
2    mgmt-1   No Carrier -                       -
3    40gige-1 Up         groupnet0.subnet1.pool1 172.16.1.13
3    40gige-2 No Carrier -                       -
3    mgmt-1   No Carrier -                       -
4    40gige-1 Up         groupnet0.subnet1.pool1 172.16.1.14
4    40gige-2 No Carrier -                       -
4    mgmt-1   Up         groupnet0.subnet0.pool0 10.1.1.14
-------------------------------------------------------------
Total: 12
```

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.

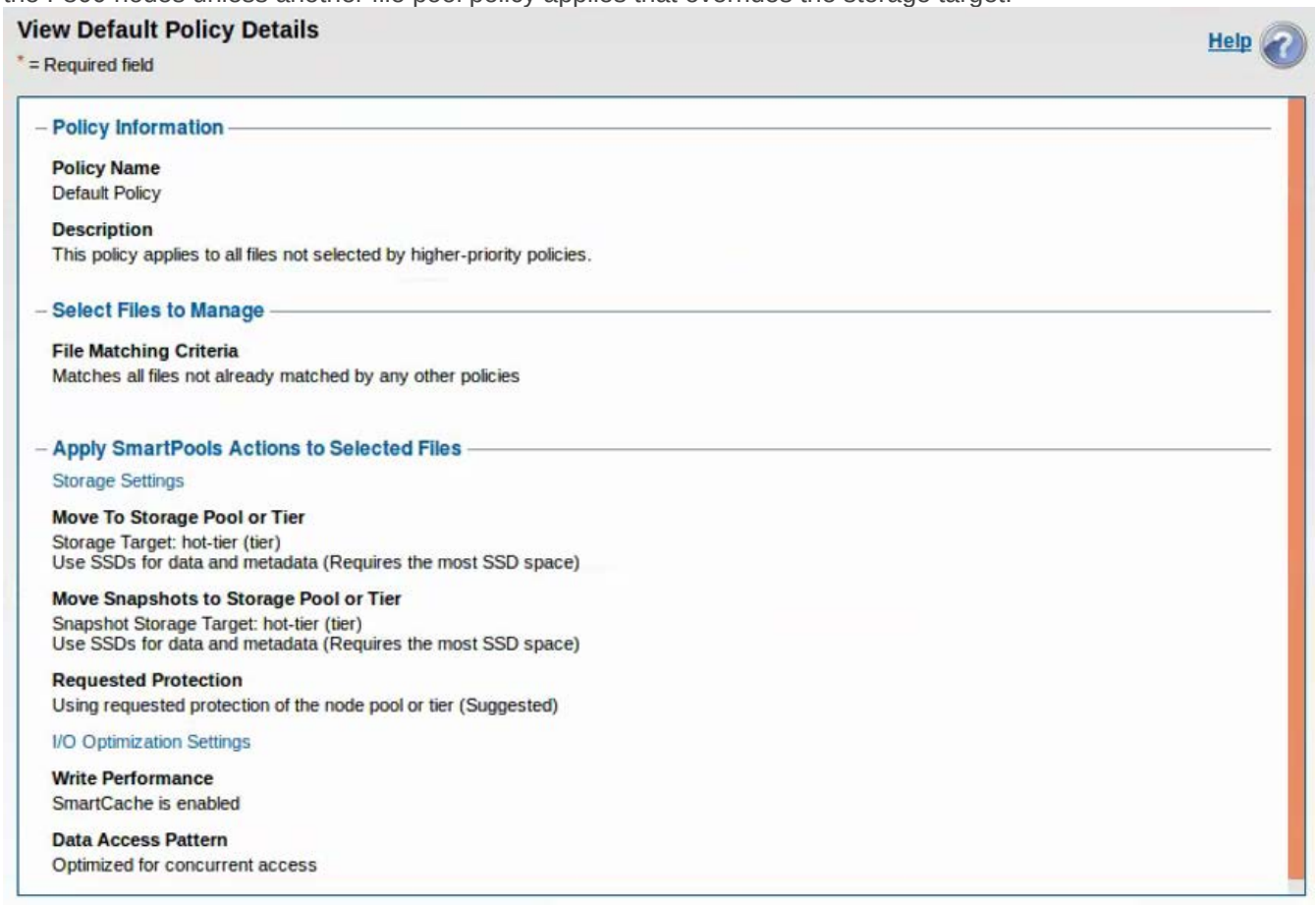
Configuring Automatic Storage Tiering

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.

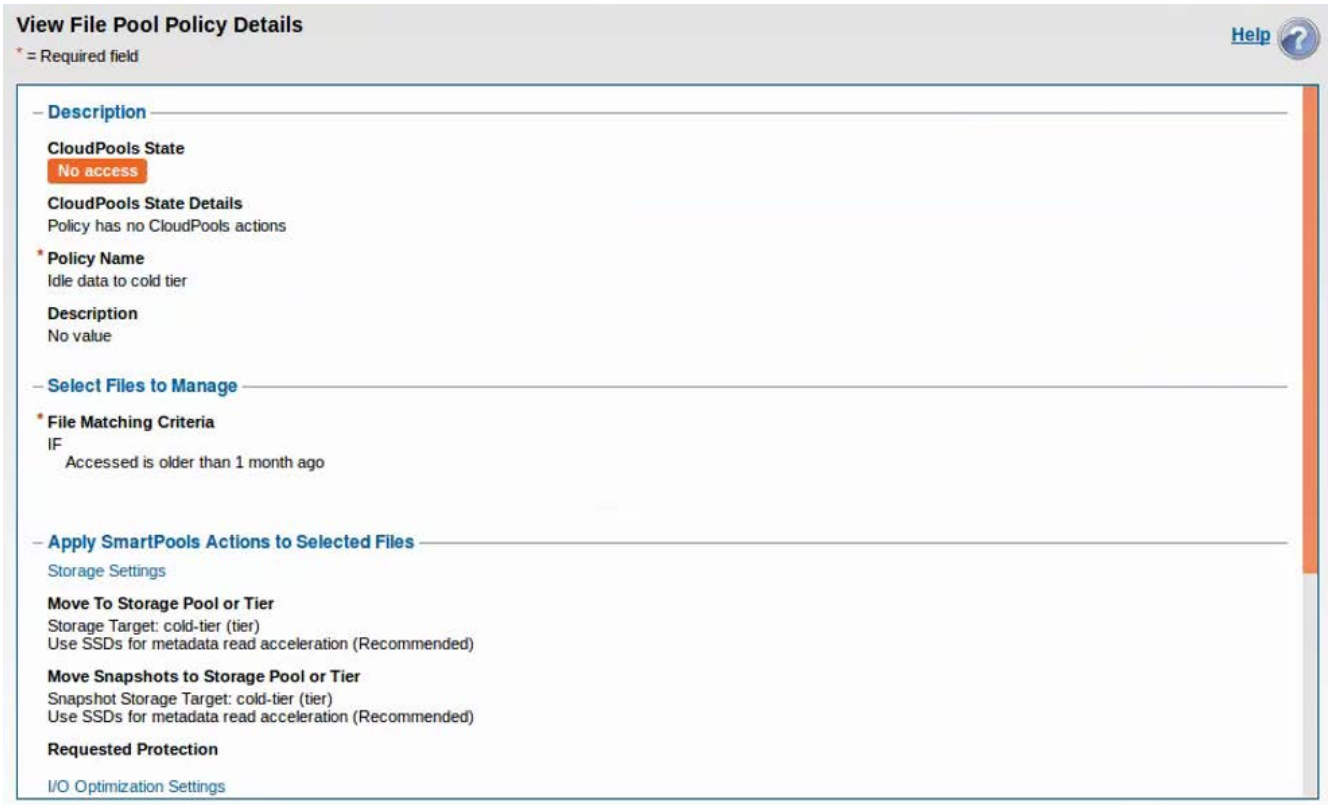
1. Build an Isilon cluster with two or more node types. For example, F800 all-flash nodes for a hot tier and H500 hybrid nodes for a cold tier.
2. Obtain and install the Isilon SmartPools license.
3. Enable Isilon access time tracking with a precision of 1 day. In the web administration interface, click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



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



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 the *isi get* command to confirm that a file is stored in the hot tier. Note that 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)
```

3. 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
```

4. 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
```

5. 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
```

6. 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)
```

7. Run the training benchmark as described in the previous section. Be sure to run at least 4 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.
8. 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
```
9. 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 DSS 8440

Although the DSS 8440 server as tested had four 40 GbE NICs, only one was used, as it provided sufficient bandwidth for all benchmarks performed.

Below is an example of the file */etc/network/interfaces*.

```
auto lo
iface lo inet loopback
# The primary network interface
auto eno3
iface eno3 inet dhcp
auto enp31s0f0
iface enp31s0f0 inet static
    netmask 255.255.255.0
    address 172.16.1.1
auto enp31s0f1
iface enp31s0f1 inet static
    netmask 255.255.255.0
    address 172.16.1.2
```

If you are installing 8 GPUs in a DSS 8440 server, be sure to install the GPUs in the inner 8 PCIe slots. This will distribute the PCIe load evenly across the PCIe switches. If you are attempting to use only 8 GPUs in a server with 10 GPUs installed, care should be taken to ensure that only the inner 8 GPUs are utilized.

DELL EMC NETWORKING S5232 SWITCH

The S5232 switch was used in its default configuration. Ports were configured for 40 GbE.

Below are the most important configuration commands.

```
ip vrf default
!
interface breakout 1/1/21 map 40g-1x
interface breakout 1/1/22 map 40g-1x
interface breakout 1/1/23 map 40g-1x
interface breakout 1/1/24 map 40g-1x
interface breakout 1/1/25 map 40g-1x
interface breakout 1/1/26 map 40g-1x
interface breakout 1/1/27 map 40g-1x
interface breakout 1/1/28 map 40g-1x
interface breakout 1/1/29 map 40g-1x
interface breakout 1/1/30 map 40g-1x
interface breakout 1/1/31 map 40g-1x
interface breakout 1/1/32 map 40g-1x
!
interface vlan1
 no shutdown
!
```

```
interface ethernet1/1/21:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/22:1
          no shutdown
 switchport access vlan 1
!
interface ethernet1/1/23:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/24:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/25:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/26:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/27:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/28:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/29:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/30:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/31:1
 no shutdown
 switchport access vlan 1
!
interface ethernet1/1/32:1
 no shutdown
 switchport access vlan 1
```

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 172.16.1.11:/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 the 40 Gbps interface on each Isilon node, 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 DSS 8440 server to mount to each of the 4 Isilon interfaces.

```
common_mount_opts=\
```

```
"-o rsize=524288,wsize=524288,nolock,soft,timeo=50,retrans=1,proto=tcp"
mount -t nfs 172.16.1.11:/ifs ${common_mount_opts} /mnt/isilon1
mount -t nfs 172.16.1.12:/ifs ${common_mount_opts} /mnt/isilon2
mount -t nfs 172.16.1.13:/ifs ${common_mount_opts} /mnt/isilon3
mount -t nfs 172.16.1.14:/ifs ${common_mount_opts} /mnt/isilon4
```

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.

In such a mounting scheme, you must ensure that the application evenly uses the data in all mount points. 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 DSS 8440 server, 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 DSS 8440 server RAM, consider using fewer mounts to allow your entire dataset to be cached.

Appendix B TensorFlow CNN Benchmarks setup

This section shows the procedure to prepare a cluster of DSS 8440 servers to run the TensorFlow CNN Benchmarks. It uses several Bash and Python scripts that are available at https://github.com/claudiofahey/tensorflow-benchmark-util.

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_records.sh was used. To create the 22.5 TB dataset, consisting of larger resized images, the script resize_tfrecords.sh was used to resize the images and then the resulting dataset was copied 13 times using expand_records2.sh.

OBTAIN THE TENSORFLOW BENCHMARKS

The TensorFlow Benchmarks 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 85f0b6b
```

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/6cdfc0c.

To allow the TensorFlow Benchmarks to be modified quickly, the scripts are not included (burned in) to the Docker image.

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. On each DSS 8440 server, 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 DSS 8440 servers via SSH on port 2222.

First, a custom Docker image is created using the following Dockerfile. Note that the container version may also be 18.09-py3, depending on the version of TensorFlow that you want to test.

```
FROM nvcr.io/nvidia/tensorflow:19.03-py3
```

```
# Install SSH.
RUN apt-get update && apt-get install -y --no-install-recommends \
openssh-client \
openssh-server \
&& \
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/*
EXPOSE 2222
```

As you can see, this Dockerfile is based on the NVIDIA GPU Cloud (NGC) TensorFlow image. You must create an account on NGC (there is no charge for this account) and configure Docker to login using this account using the ***docker login nvcr.io*** command.

Once logged in, 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:19.03-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 DSS 8440 servers. 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:19.03-py3-custom.
```

You are now ready to start the containers on all DSS 8440 servers. Repeat this command for each DSS 8440 server, 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:19.03-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 DSS 8440 servers as the master 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 dss8440-1 hostname
```

Next, test that MPI can launch processes across all DSS 8440 servers.

```
mpirun --allow-run-as-root -np 2 -H dss8440-1 -H dss8440-2 hostname
```

To stop the containers and all processes within them, run the following command on each DSS 8440 server

```
docker stop tf
```

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

Appendix C 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 more details about InsightIQ, refer to the Isilon InsightIQ User Guide.

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___                                       _____OneFS Stats_____
user              1.4%                                      In          73.81 kB/s
system           72.5%                                      Out          8.96 GB/s
idle             26.1%                                      Total        8.96 GB/s

____Network Input____        ___Network Output____        ______Disk I/O_______
MB/s            12.64        MB/s           9868.11       Disk   272334.03 iops
```

```
Pkt/s       150368.20        Pkt/s      6787182.27        Read       11.73 GB/s
Errors/s         0.00        Errors/s         0.00        Write      99.63 MB/s
```

Appendix D 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 your head node (or first compute node), create a hosts file containing one host name (or IP address) per client. For example:

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

IPERF

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

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

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

FIO

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

Mount to Isilon. Each client will mount to a different Isilon IP address.

```
cat hosts | xargs -i -P 0 ssh root@{} mkdir -p /mnt/isilon
cat hosts | xargs -i -P 0 ssh root@{} umount /mnt/isilon
ssh root@server1.example.com mount -t nfs 10.1.1.1:/ifs \
-o rsize=524288,wsize=524288 /mnt/isilon
ssh root@server2.example.com mount -t nfs 10.1.1.2:/ifs \
-o rsize=524288,wsize=524288 /mnt/isilon
```

Install FIO servers.

```
cat hosts | xargs -i -P 0 ssh root@{} yum -y install epel-release
cat hosts | xargs -i -P 0 ssh root@{} yum -y install fio
cat hosts | xargs -i ssh root@{} fio –version
```

Start FIO servers.

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

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/isilon/tmp/fio
time_based=1
```

```
runtime=600
ramp_time=60
ioengine=libaio
numjobs=8
create_serialize=0
iodepth=32
kb_base=1000
[job1]
rw=read
nrfiles=30
size=64GiB
bs=1024KiB
direct=1
sync=0
rate_iops=83
```

Run the FIO job.

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

Appendix E Collecting system metrics with ELK

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. This section shows how to do this with Elasticsearch, Kibana, Metricbeat, and NVIDIA GPU Beat. This suite of tools from Elastic is often referred to as the ELK stack, however, Logstash was not used.

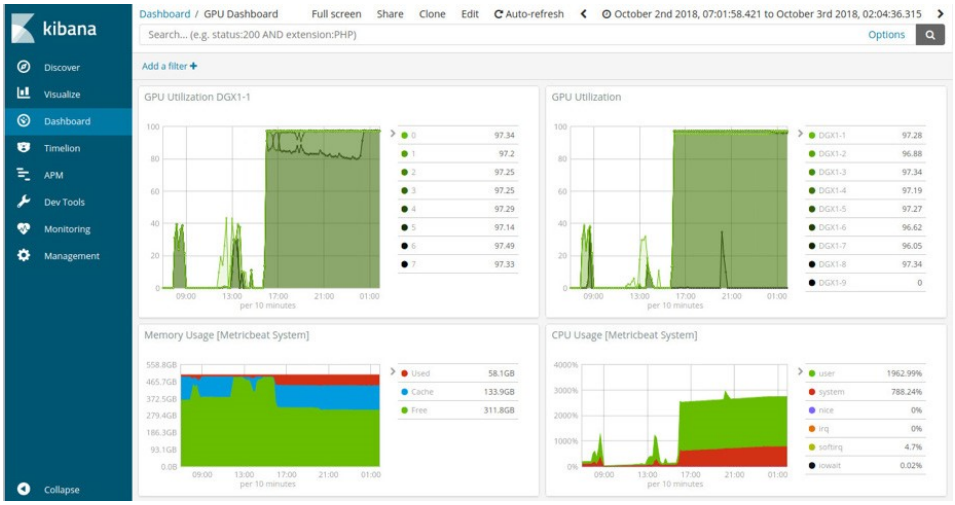


Figure 12: A custom Kibana dashboard showing GPU, CPU, and memory utilization. Data is collected by Metricbeat and NVIDIA GPU Beat.

A single-node deployment of Elasticsearch and Kibana can be brought up in a few minutes using the Docker Compose file ***elk/docker-compose.yml*** in the ***tensorflow-benchmark-util*** repository.

```
cd tensorflow-benchmark-util/elk
docker-compose up –d
```

Next, Metricbeat must be started on all DSS 8440 servers. Edit the file metricbeat/hosts to contain the FQDN for all DSS 8440 servers. Edit the file ***metricbeat/etc/metricbeat/metricbeat.yml*** to contain the IP address or FQDN for the server running Elasticsearch. Then run the ***script start_metricbeats.sh*****.** This will SSH into each host and start the Metricbeat container.

For NVIDIA GPU Beat, repeat the above steps but in the nvidiagpubeat directory. Note that the version of NVIDIA GPU Beat dated September 25, 2018 has a resource leak that causes resource exhaustion after about 24 hours. Be sure to restart the containers every few hours to avoid this.

Open your browser to port 5601 of the server running Kibana. Import the customized Kibana dashboards and visualizations in ***elk/*kibana.json.*** Finally, open the dashboard named **GPU Dashboard.**

Appendix F Tips

1. When MPI TensorFlow applications are terminated by the user with Control-C or by application errors, some processes may not terminate completely. This may result in subsequent executions getting a GPU out-of-memory (OOM) error. This condition can be confirmed by running nvidia-smi and checking the used memory. To fix this, simply stop and restart the TensorFlow containers.
2. Avoid significant amounts of network traffic on the inter-switch links. Since there are no VLANs set up, this can be done accidently if NICs are not assigned to the proper IP subnets. Check the interface packet counters on the data switches before and after a training run to confirm this.

References

Name	Link
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
DSS 8440	https://blog.dellemc.com/en-us/dell-emc-dss-8440-dynamic-machine-learning-server/
Elasticsearch, Kibana	https://www.elastic.co/
Horovod	https://github.com/uber/horovod
ImageNet	http://www.image-net.org/challenges/LSVRC/2012/
Isilon InsightIQ	https://www.emc.com/collateral/TechnicalDocument/docu65870.pdf
Isilon Storage Tiering	https://www.dellemc.com/resources/en-us/asset/white-papers/products/storage/h8321-wp-smartpools-storage-tiering.pdf
Isilon OneFS Best Practices	https://www.emc.com/collateral/white-papers/h16857-wp-onefs-best-practices.pdf
Isilon OneFS Technical Overview	https://www.emc.com/collateral/hardware/white-papers/h10719-isilon-onefs-technicaloverview-wp.pdf
NVIDIA GPU Cloud (NGC) TensorFlow image	https://ngc.nvidia.com/registry/nvidia-tensorflow
NVIDIA mixed precision training	https://docs.NVIDIA.com/deeplearning/sdk/mixed-precision-training/index.html
NVIDIA Tesla V100	https://www.nvidia.com/en-us/data-center/tesla-v100/
S5232 Switch	https://www.dell.com/en-us/work/shop/povw/networking-s-series-25-100gbe
Storage Tiering with Dell EMC Isilon SmartPools	https://www.dellemc.com/resources/en-us/asset/white-papers/products/storage/h8321-wp-smartpools-storage-tiering.pdf
TensorFlow	https://github.com/tensorflow/tensorflow
TensorFlow Benchmarks	https://www.tensorflow.org/performance/benchmarks
TensorFlow Benchmark Utilities	https://github.com/claudiofahey/tensorflow-benchmark-util
TensorFlow Inception	https://github.com/tensorflow/models/tree/master/research/inception