HAL: Computer System for Scalable Deep Learning

Volodymyr Kindratenko
National Center for Supercomputing Applications, UIUC
Urbana, IL
kindrtnk@illinois.edu

Dawei Mu
National Center for Supercomputing Applications, UIUC
Urbana, IL
dmu@illinois.edu

Yan Zhan
National Center for Supercomputing Applications, UIUC
Urbana, IL
yanzhan2@illinois.edu

John Maloney
National Center for Supercomputing Applications, UIUC
Urbana, IL
malone12@illinois.edu

Sayed Hadi Hashemi
National Center for Supercomputing Applications, UIUC
Urbana, IL
hashemi3@illinois.edu

Benjamin Rabe
National Center for Supercomputing Applications, UIUC
Urbana, IL
brabe2@illinois.edu

Ke Xu
National Center for Supercomputing Applications, UIUC
Urbana, IL
kexu6@illinois.edu

Roy Campbell
National Center for Supercomputing Applications, UIUC
Urbana, IL
rhc@illinois.edu

Jian Peng
National Center for Supercomputing Applications, UIUC
Urbana, IL
jianpeng@illinois.edu

William Gropp
National Center for Supercomputing Applications, UIUC
Urbana, IL
wgropp@illinois.edu

ABSTRACT

We describe the design, deployment and operation of a computer system built to efficiently run deep learning frameworks. The system consists of 16 IBM POWER9 servers with 4 NVIDIA V100 GPUs each, interconnected with Mellanox EDR InfiniBand fabric, and a DDN all-flash storage array. The system is tailored towards efficient execution of the IBM Watson Machine Learning enterprise software stack that combines popular open-source deep learning frameworks. We build a custom management software stack to enable an efficient use of the system by a diverse community of users and provide guides and recipes for running deep learning workloads at scale utilizing all available GPUs. We demonstrate scaling of a `PyTorch` and `TensorFlow` based deep neural networks to produce state-of-the-art performance results.

CCS CONCEPTS

• **Computer systems organization → Special purpose systems**; • **Computing methodologies → Machine learning**; • **Social and professional topics → System management**.

KEYWORDS

deep learning, cluster architecture, high-performance computing

ACM Reference Format:
Volodymyr Kindratenko, Dawei Mu, Yan Zhan, John Maloney, Sayed Hadi Hashemi, Benjamin Rabe, Ke Xu, Roy Campbell, Jian Peng, and William Gropp. 2020. HAL: Computer System for Scalable Deep Learning. In *Practice and Experience in Advanced Research Computing (PEARC '20), July 26–30, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3311790.3396649

1 INTRODUCTION

In 2017, National Center for Supercomputing Applications (NCSA) was funded by the National Science Foundation's (NSF) Major Research Instrumentation (MRI) program to develop and deploy a computational "instrument" for supporting deep learning (DL) applications at scale[1]. The main motivation for building such an system was an apparent lack of sufficient computational resources on the University campus designated to support a growing number of researchers applying DL methodology in their work. We surveyed the campus research community and identified over 30 faculty actively applying DL who struggled to find adequate computing resources to train deep neural networks (DNNs). A typical mode of operation was to use a student-managed workstation outfitted with one or two consumer-grade NVIDIA GPUs running a sub-optimal software stack. These resources were inadequate as even simple networks of any practical use required days or weeks of time to train.

[1]https://www.nsf.gov/awardsearch/showAward?AWD_ID=1725729

The newly designed system, called the Hardware Accelerated Learning (HAL) cluster[2], encompassed the latest hardware and software components to provide a highly optimized resource that supports multiple users and scales training of many DL models to 64 GPUs across 16 compute nodes. It became operational in March 2019 and has become a major enabling technology for numerous research groups on campus.

In this paper, we describe the design, deployment, and operation of HAL and provide benchmarks to demonstrate system scalability and usability. We also describe the software components developed by the team to manage and monitor the system.

2 SYSTEM REQUIREMENTS

In 2016, we conducted a user survey to understand the landscape of computing requirements for DL. A majority of users on campus used `TensorFlow`[3] (over 70%) and `Caffe`[4] (over 20%) as the DL framework of choice. Also, the datasets used for model training typically ranged from several gigabytes to hundreds of gigabytes in size. The community was concerned that it was beyond an individual researcher's ability to acquire and maintain a system consisting of more than just a few GPU workstations, and such resources would still not be sufficient to scale their workloads. Many foresaw a need for a system to run DL workloads with 32 or more GPUs and datasets in a terabyte range, yet almost none of those surveyed knew what it would take to scale their models to such a size and how to run distributed DL workloads. Therefore, we considered performance, scalability and usability aspects when designing the system.

2.1 Performance Requirements

Many modern DL frameworks have been optimized to make use of the latest NVIDIA GPUs. At the time of planning for the system, P100 GPUs[5] were state-of-the-art, and V100 GPUs[6] were just announced by NVIDIA. V100 GPUs promised to significantly increase DL training performance due to the specially designed tensor cores. Therefore, they were deemed the most desirable feature to have in the newly designed system. Also, scaling DL frameworks to multiple GPUs would require a high-performance interconnect within a node as well as across multiple nodes. The NVIDIA DGX-1 system[7] utilized NVLink[8] interconnects to deliver cross-GPU bandwidth well in excess of standard Gen. 3 PCIe x16 interfaces. Therefore, having NVLink-interconnected GPUs was deemed to be the next most important requirement. The CPU performance was also important; the limiting factor was not the number of CPU cores, but rather the system memory bandwidth to sustain these cores.

Many DL workloads have storage needs for training sets on the order of terabytes. The most important requirement for distributed DL is to be able to feed all computational units with the data so the storage system does not become a single point of contention. Thus, the system must have storage and node interconnects that can support simultaneous data processing by all GPUs without degrading performance of individual GPUs.

2.2 Usability Requirements

Supporting a DL user community requires not only enabling a diverse set of relevant tools and frameworks, but also enabling diverse usage modalities. Traditional HPC systems use the concept of job submission and resource scheduling to execute the user application on the system. Users typically access the system via `ssh` and interact with it through the command line interface. On the other hand, many DL communities heavily utilize interactive applications, using tools such as `Jupyter Notebook` through web-based interfaces. These tools work well in a cloud environment where computational resources are unlimited; however, their use on a fixed-size shared hardware is non-trivial. Therefore, it was deemed necessary to provide both types of system access while maintaining an efficient resource allocation and sharing.

Frameworks, such as `TensorFlow`, are frequently updated and the user community is eager to start using new features the moment they are introduced. On the other hand, applications developed with a particular version of `Python` or 3rd party libraries still need to run. This requirement can be satisfied by using a modular approach to provide execution environments via modules, containers, and isolation of Python environments within the user space.

Other user requirements included the having the ability to easily transfer data to and from the system, having access to pre-loaded "standard" datasets (such as `ImageNet`), and being able to allocate an arbitrary number of nodes for exclusive, prolonged use.

3 HAL SYSTEM DESIGN

Based on the performance and usability requirements, we conducted a survey of technology trends. We considered both x86 and POWER9 architectures, built a prototype x86 node outfitted with NVIDIA V100 GPUs, and acquired an IBM POWER9 AC922 server [2] with NVLink 2.0 interconnected V100 GPUs. We ran various DL benchmarks, such as those provided for `TensorFlow`, and compared our results with those reported in the literature for other systems. We developed tools to benchmark storage options and worked with vendors to evaluate several storage offerings. Based on these evaluation and prototyping efforts, we selected an IBM POWER9 based solution with NVIDIA GPUs as the compute core, Mellanox EDR InfiniBand interconnect, and DDN storage system. Details of this design are provided below.

3.1 Hardware Architecture

The overall hardware architecture is shown in Figure 1. The system is composed of one login node, two management nodes, two storage nodes, one test/development node, and 16 compute nodes. The nodes are interconnected with EDR InfiniBand network for storage and inter-node communication and a Gigabit Ethernet for management and access. Two DDN flash arrays serve as a parallel file system for the cluster.

The main management node is an IBM LC921 server[9] with two IBM POWER9 CPUs. This node is connected to all internal networks and performs administrative functions including remote

[2]https://wiki.ncsa.illinois.edu/display/ISL20/HAL+cluster
[3]https://www.tensorflow.org
[4]https://caffe.berkeleyvision.org
[5]https://www.nvidia.com/en-us/data-center/tesla-p100
[6]https://www.nvidia.com/en-us/data-center/v100
[7]https://www.nvidia.com/en-us/data-center/dgx-1
[8]https://www.nvidia.com/en-us/data-center/nvlink
[9]https://www.ibm.com/us-en/marketplace/power-system-lc921-and-lc922

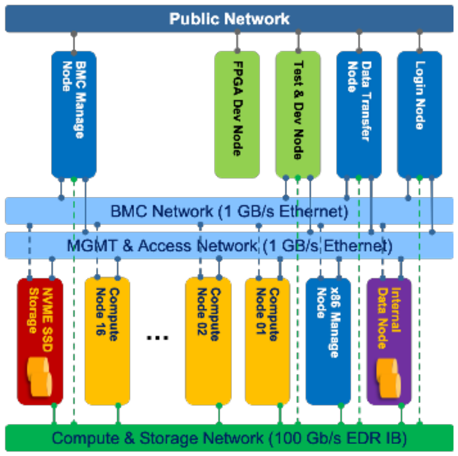

Figure 1: The overall hardware architecture of HAL system.

management via BMC, OS image storage, and deployment; it also runs databases for metrics and job accounting. The main OS resides on a pair of SSDs in RAID1 configuration, while OS images and metrics are stored on another RAID1 array. The second management node is an x86 server used to run auxiliary services that are currently incompatible with the IBM POWER9 architecture, such as `Puppet`[10] and the `Open OnDemand`[11] web interface, which will be described later.

The login node is also an IBM LC921 server. This node serves as the main interface between users and the computing resources on the system. The `Slurm` scheduler[12] runs on this node and allows users to submit jobs for execution. To simplify the development process for users, the software environment on this node is kept in sync with compute nodes.

One of the storage nodes (Internal Data node in Figure 1) is an IBM LC922 server with a 12-drive RAID60 array. This node was initially designed to be a stop-gap solution for cluster-wide shared storage while we conducted an evaluation for a permanent storage solution. Another storage node (Data Transfer Node in Figure 1) is an x86 server that holds 36 hardware drives. At present, these nodes are connected to the rest of the cluster via NFS and are dedicated for miscellaneous internal use.

For compute nodes, we selected IBM AC922 8335-GTH servers [2]. Each of these servers contains two 20-core IBM POWER9 CPUs, 256 GB of DDR4 RAM, and four NVIDIA V100 GPUs with 16 GB of HBM2 RAM each and NVLink 2.0 interconnect. These servers were chosen because of their unique expansion bus which provides PCIe 4.0 connectivity and allows the CPUs to utilize the NVLink fabric. A dual-port Mellanox ConnectX-5 InfiniBand EDR 100Gb/s adapter[13] is used to provide high-performance communication. These InfiniBand adapters also support GPUDirect[14], which allows direct communication between GPUs themselves and between GPUs and the network adapter. Without going through CPUs, performances is improved for multi-GPU and multi-node workloads.

Two DDN GS400NVE Flash Arrays serve as the primary storage solution for the entire cluster. These storage nodes provide 224 TB of usable capacity of NVMe SSD-based storage which are capable of a peak cluster-aggregate bandwidth of over 90GB/s. These nodes use IBM's Spectrum Scale[15] file system.

Total power consumption, as measured during the all-system distributed DL training, is around 25 kW.

Storage Selection & Tuning. The hardware to back the shared storage component of HAL was a critical choice to ensure that the highly optimized compute hardware could be fed with data at a sufficient rate. Many DL workloads display small random read I/O patterns as they train across a data set. An all-flash file system design was prime candidate to satisfy these workloads in the early part of the decision-making process, with the goal being to greatly reduce or eliminate the storage sub-system as a bottleneck during training runs at scale.

We considered solutions from multiple vendors and received performance numbers from each vendor on three benchmarks with their proposed solution. Those benchmarks were `IOR`[16], `mdtest`[17], and a custom benchmark we developed based on MPI-coordinated FIO that mimicked the behavior of several DL training workloads, DIOT[18]. For the `IOR` and `mdtest` sections of the benchmarks, vendors provided us with the numbers that reflected the best case performance of their solutions. After reviewing the benchmark results, a key distinction emerged between the NFS-based and Spectrum Scale-based solutions. There was a much higher standard deviation between the performance of different ranks with the NFS-based solutions vs the Spectrum Scale solutions. This indicated that there was noticeable variability between the I/O performance each thread was receiving from the file system during the benchmark runs.

Figures 2 and 3 show results obtained using our own benchmark, DIOT. The benchmark mimics I/O patterns of several DL workloads and measures bandwidth and IOPs of the storage system. Figures 2 and 3 show aggregate bandwidth and IOPs for four storage systems to show relative performance of each. Here, "NVME single node" refers to a PCIe NVME drive installed in one of the HAL nodes, "NFS sum" refers to the performance obtained for the IBM LC922 storage server mounted on the cluster over NFS protocol, and "DDN 4 nodes sum" refers to results obtained with the DDN solution using 4 client nodes. For reference, we also run DIOT benchmark on our campus cluster system[19] that uses a much larger, HDD-based DDN storage system. These results show the NVME-based DDN system performance and consistency across many DL workloads.

After narrowing the solution candidates based on the benchmark results, we also considered Cost per Usable TB to maximize the amount of space we could provide users with. In the end, we procured two DDN GS400NVE units, each with 21 7.68TB Samsung PM1725b NVME SSDs. Combined, the two units connect into the

[10]https://puppet.com
[11]https://osc.github.io/ood-documentation/master
[12]https://slurm.schedmd.com
[13]https://www.ibm.com/support/knowledgecenter/POWER9/p9hcd/fcec64.htm
[14]https://developer.nvidia.com/gpudirect

[15]https://www.ibm.com/us-en/marketplace/scale-out-file-and-object-storage
[16]https://github.com/hpc/ior
[17]https://sourceforge.net/projects/mdtest
[18]https://github.com/xldrx/diot
[19]https://campuscluster.illinois.edu/

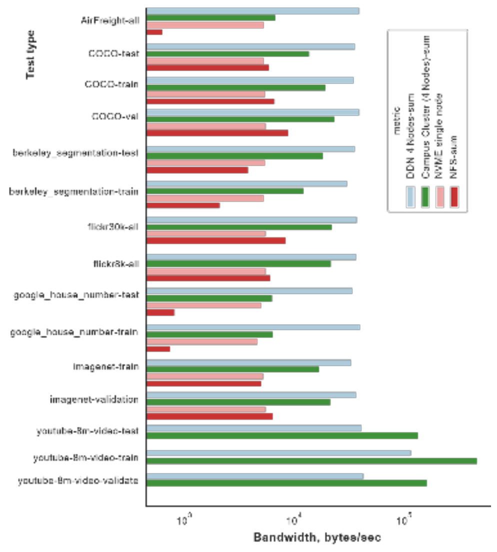

Figure 2: Bandwidth test with DIOT benchmark suite

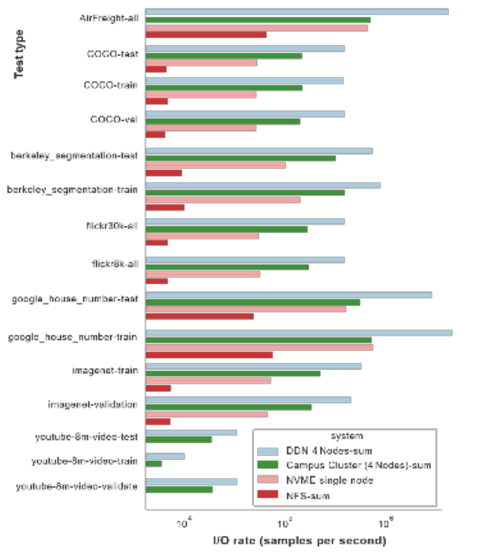

Figure 3: IOPS test with DIOT benchmark suite

storage fabric with 8 EDR cables, providing 800Gb/s of network bandwidth to the storage.

Tuning the storage subsystem required a balance between leveraging ample bandwidth and IOPs potential of the hardware. Due to the advanced sub-block allocation capability of Spectrum Scale, we did not have to worry about file space efficiency loss if we wanted to choose a larger block size. However, in the end we chose a smaller block size of 1MB due to the file sizes we were seeing across the user workloads. On the embedded storage servers, adjustments were made to `nsdbuffspace` and the `nsdMaxWorkerThreads`, `workerThreads`, and `nsdMinWorkerThreads` parameters were increased as well as the file system's log size. Other Spectrum Scale adjustments included increasing the `maxFilesToCache` and `maxStatCache` parameters across the cluster to improve metadata performance.

On the underlying hardware itself, we conducted tests in coordination with DDN to tune the RAID devices by setting appropriate chunk and stripe sizes. VRC and SGC values were also tuned to align I/O between the file system and the supporting hardware's Stripe Group Size to maximize performance.

Over the life of the file system, in addition to detailed bandwidth, iops, and metadata statistics, measurements of the time it takes to list home directories and stat a file on the file system have been captured at 60 second intervals to instrument the responsiveness user's feel on the system. The mean time to list all home directories has been 7ms with a standard deviation of just 17.8% and the mean time to stat a file on the file system being 2ms with a standard deviation of just 18.6%.

3.2 Software Stack

CentOS 7.7 LE ALT for POWER9 (ppc64le – powerpc 64-bit little endian)[20] is used as the OS on our IBM AC922 servers and all supporting nodes. IBM Watson Machine Learning Community Edition (`WMLCE-1.7.0`) [10] is an enterprise software distribution that combines popular open-source deep learning frameworks and efficient AI development tools optimized for the IBM POWER9 architecture. It includes `Caffe`, `Tensorflow` and `Pytorch`[21]. We support both `python` 2.7 and 3.6/3.7 versions of `WMLCE` via Environment Modules Lmod[22]. Other software components include `NVIDIA CUDA` 10.2 tools[23], PGI compiler[24], the IBM Advance toolchain for Linux on Power[25] set of open source compilers, runtime libraries, and development tools, `Jupyter Notebook` and `Jupyter Lab`[26], `Tensorboard`[27] and `H2O`[28].

Our users come from a variety of fields with varying degrees of technical expertise. Many have little to no experience with command line utilities common on HPC systems, so we have strove to develop simplified interfaces and utilities as well as tutorials and guides. These utilities developed in house include `Slurm Wrapper Suite`, `Open OnDemand` portal, and monitoring services.

3.2.1 *Slurm Wrapper Suite.*

The Slurm Wrapper Suite (`SWSuite`) is designed to simplify the use of the `Slurm` resource allocation and job submission utility. `SWSuite` automatically generates appropriate resource allocation parameters for `Slurm`, minimizing the required input from the users and ensuring consistency and homogeneity of resource utilization. There are three main programs in `SWSuite`: `swrun`, `swbatch`, and `swqueue`.

To submit a job, a user requests a virtual partition defined by `SWSuite`. Several such partitions are provided, namely gpux1, gpux2,

[20] https://wiki.centos.org/SpecialInterestGroup/AltArch/ppc64le
[21] https://pytorch.org
[22] https://lmod.readthedocs.io
[23] https://developer.nvidia.com/cuda-zone
[24] https://www.pgroup.com
[25] https://developer.ibm.com/linuxonpower/advance-toolchain
[26] https://jupyter.org
[27] https://www.tensorflow.org/tensorboard
[28] https://www.h2o.ai/products/h2o

... for using 1, 2 or more GPUs and cpun1, cpun2, ... for running CPU only tasks. Based on the selected partition, SWSuite computes the remaining Slurm run-time parameters. The code snippet below shows srun parameters versus swrun parameters needed to accomplish the same task.

```
srun --partition=gpu --time=24:00:00 --nodes=1 --ntasks-per-node=160 \
     --sockets-per-node=2 --cores-per-socket=20 --threads-per-core=4 \
     --mem-per-cpu=1200 --wait=0 --export=ALL --gres=gpu:v100:4 --pty /bin/bash
```

```
swrun -p gpux4 -c 40 -t 24
```

Similarly, the batch submission script itself can use these partitions to simplify the specification of job parameters. swrun and swbatch are designed to enhance the user experience of Slurm rather than replace its original functionality.

The swqueue utility provides a visual snapshot of the current job status on the system within the command line terminal, as shown in Figure 4. It parses the output of squeue in order to generate this visualization.

3.2.2 HAL OnDemand. Our HAL OnDemand portal is based on the Open OnDemand project [6], which is an NSF-funded open-source HPC portal developed at the Ohio Supercomputer Center. The goal of Open OnDemand is to provide an easy way for system administrators to provide web-based access to their HPC resources, including a plugin-free web experience, easy file management, command-line shell access, job management, graphical desktop environments, and desktop applications.

One of the advantages of using HAL OnDemand is that only a modern web browser is needed on the user's side to fully utilize HAL's resources without the need for a traditional terminal access. HAL OnDemand provides the graphical user interface that allows new users not familiar with command-line shell, which is still available within the web browser, to edit and submit computational tasks, as shown in Figure 5. Moreover, HAL OnDemand provides an easy to use solution for starting interactive applications, as shown in Figure 6. Currently supported HAL interactive applications include Jupyter Notebook, Jupyter Lab, Tensorboard and H2O. We also provide a Singularity container option for Jupyter Notebook that implements a different version of TensorFlow. Centrally managed LDAP directory based user authentication is integrated with HAL OnDemand portal, providing user login.

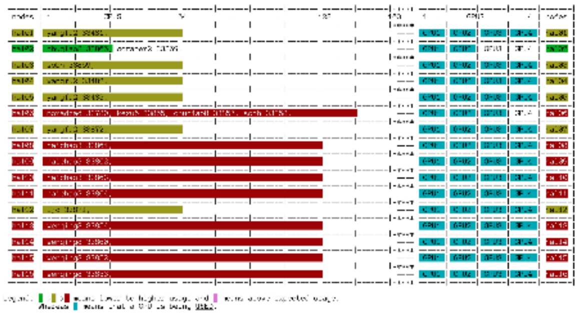

Figure 4: Output of the swqueue command

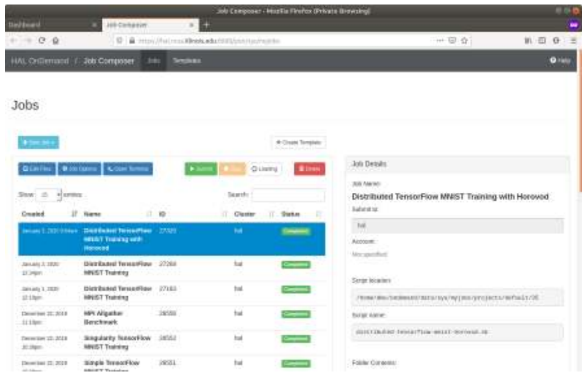

Figure 5: HAL OnDemand job composer

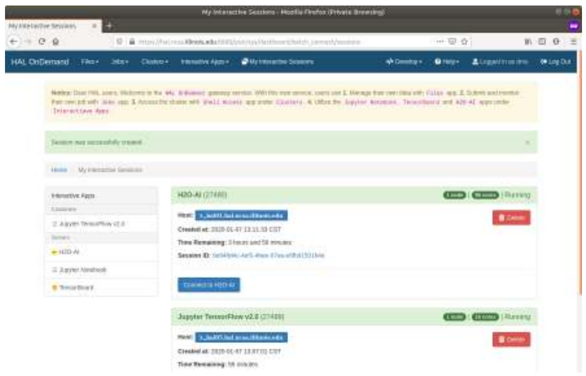

Figure 6: HAL OnDemand interactive sessions

3.2.3 Monitoring Stack. On modern HPC systems, a graphical monitoring interface is often implemented to help both users and administrators understand the utilization of the system. We employ a Telegraf[29]-InfluxDB[30]-Grafana[31] stack for our monitoring solution. On every node in the cluster, a Telegraf agent collects metrics about the system and sends them to the central InfluxDB server on the management node, which then stores the metrics into a time-series database. The Grafana server is set up on a virtual machine outside of the cluster, so it can stay up during downtimes and allow users to check the availability of the system.

In addition to the built-in Telegraf plug-ins, we developed a collection of custom scripts to add metrics that are specific to our system. This includes a script to communicate with IBM's ibm-crassd telemetry service to collect metrics from the BMC controllers on the compute nodes and inject them into Telegraf, providing hardware measurements such as temperature and power consumption; and a script to read job data from Slurm's accounting database and provide current and historical information for jobs and resource allocations. Figures 7, 8, and 9 show examples of the status data users and administrators can have access to. Also, the main dashboard shown in Figure 7 is conveniently accessible through a web

[29]https://github.com/influxdata/telegraf
[30]https://www.influxdata.com
[31]https://grafana.com

browser on a mobile phone, enabling users to get a quick view of the system status and their job status.

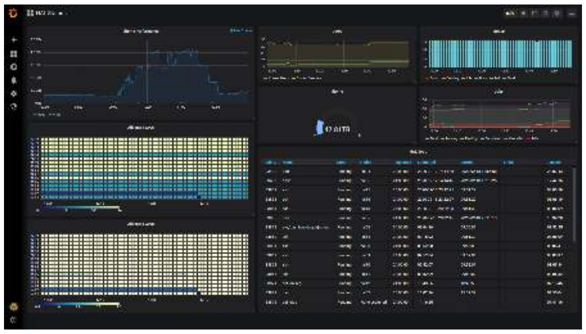
Figure 7: Main dashboard showing overall system status and job information

3.2.4 HAL System on Mobile Platforms. Through `Open OnDemand` and `Grafana Dashboard`, the HAL system can be accessed from a mobile device. Figure 10 shows examples of `Open OnDemand` and `Grafana Dashboard` interfaces. Compared to `Open OnDemand`, `Grafana Dashboard` has better compatibility with mobile platforms. As shown in Figure 11, HAL users can use their mobile web browser to check HAL system overall status and the administrator can monitor the system in real time with a hand-held device.

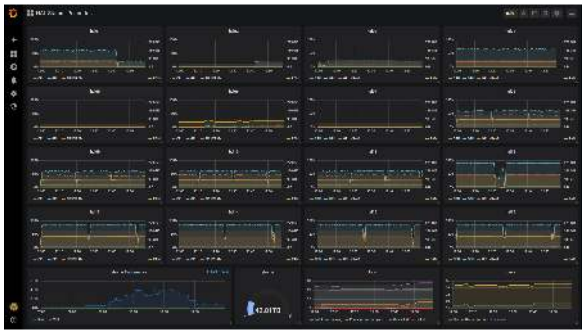
Figure 8: Dashboard showing utilization of individual nodes

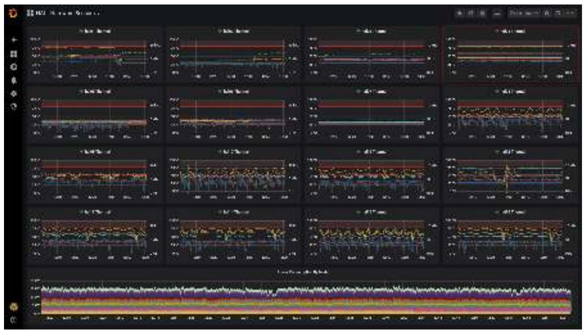
Figure 9: Dashboard for administrators to monitor thermal load and power usage

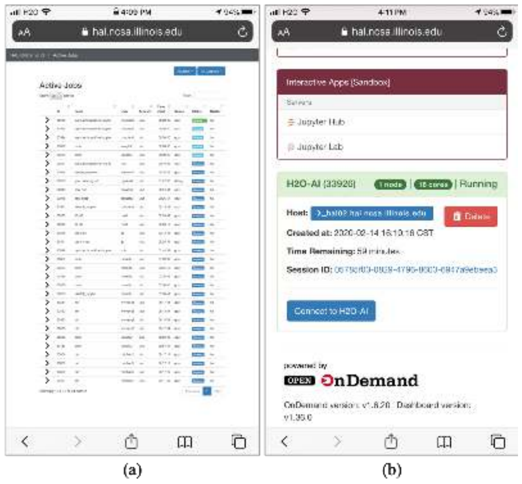

Figure 10: HAL OnDemand Service on iOS. (a) Activate Job Status. (b) H2O-AI Interactive Session.

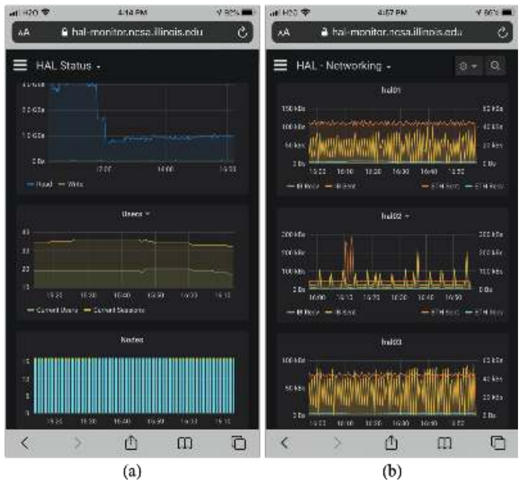

Figure 11: Grafana Dashboard on iOS. (a) Overall System Status Dashboard. (b) System Network Status Dashboard.

4 IMAGENET DISTRIBUTED MIXED-PRECISION TRAINING BENCHMARK

4.1 Implementation Details

To demonstrate the performance and scalability of our system, we conducted ImageNet[32] training by scaling ResNet-50 [5] across multiple GPUs and multiple compute nodes. We performed this benchmark with both `TensorFlow` [1] and `PyTorch` [9]. We used official implementations of ResNet-50 for both frameworks. The optimizer utilized standard momentum with m of 0.9 and a weight decay λ of 0.0001. All models were trained for 90 epochs regardless of batch sizes. We performed the learning rate scaling and gradual

[32]http://www.image-net.org

warmup mentioned in [3] to tackle training instability at early stages for large batch size. Most of our training setup was consistent with [3] and [13].

We scaled ResNet-50 from 4 GPUs on the same compute node to 64 GPUs across 16 compute nodes, doubling the number of GPUs for each run. We used a per-GPU batch size of 256 images, which is the largest batch size we can fit on our 16GB V100 GPUs with mixed-precision training. Therefore, our global batch size ranges from 1024 images to 16384 images. For both frameworks, we used the NVIDIA Collective Communication Library (NCCL)[33] as our communication backend.

For the `TensorFlow` implementation, we used the official `tf_cnn_benchmarks` repository with Horovod [11] handling distributed training. For compatibility with our `TensorFlow` version, we used the `cnn_tf_v1.14_compatible` branch of the source code, which is available on GitHub[34].

For `PyTorch`, we utilized Automatic Mixed Precision (Amp) and Distributed Data Parallel (DDP) from NVIDIA Apex[35] for mixed-precision and distributed training. An optimization level of "O2" was used for mixed-precision training to benefit from FP16 training while keeping a few parameters to be FP32. Source code of our benchmark is available on GitHub[36].

4.2 Analysis of Results

Training Time and Throughput. Figure 12 shows the amount of time taken to reach 90 epochs of training. The number of GPUs ranges from 4 GPUs to 64 GPUs. ImageNet training with ResNet-50 using 4 GPUs takes 10 hrs, 21 mins, 19 secs with TensorFlow and 11 hr 21 min 48 sec with PyTorch. With 64 GPUs across 16 compute nodes, we can train ResNet-50 in 41 mins, 43 secs with TensorFlow and 56 min, 18 sec with PyTorch, while maintaining comparable top-1 and top-5 accuracy.

Figure 13 shows the global throughput (images/sec) with respect to the number of GPUs. While throughput is effectively the inverse of training time, it more effectively shows that we were able to achieve near linear performance scaling in both frameworks.

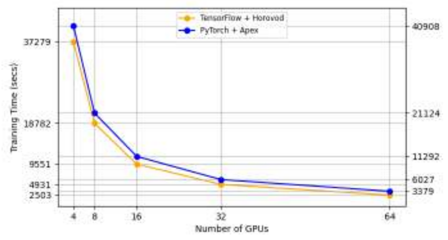

Figure 12: ImageNet ResNet-50 Training Time vs. Number of GPUs with a fixed 90 epochs training schedule.

Top1, Top5 Accuracy. We achieve distributed training speedup without the loss of accuracy. All experiments reached a top-1 validation accuracy of 76% with the exception of the 64 GPU case in the `PyTorch` benchmark, which reached 73%. This slight loss of accuracy is not surprising, as the maximum batch size in [3] was 8192, after which they experienced accuracy degradation. We included the 64 GPU case with global batch size 16384 anyway to show system-wide scalability.

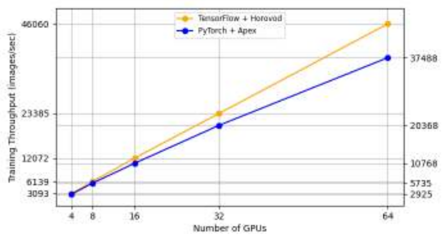

Figure 13: ImageNet ResNet-50 Training Throughput vs. Number of GPUs with a fixed 90 epochs training schedule.

Figures 14 and 15 show the top-1 and top-5 validation accuracy during the `TensorFlow` benchmark. We do notice that with larger global batch size, the training is more unstable at early stages (epoch 1-30). However, the small and large batch size training curves match closely after 30 epochs and they all peak at comparable accuracy near the end of training.

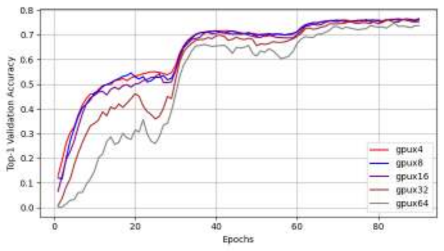

Figure 14: ImageNet ResNet-50 Top-1 Validation Accuracy.

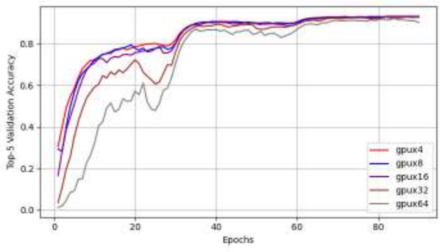

Figure 15: ImageNet ResNet-50 Top-5 Validation Accuracy.

I/O Bandwidth. Figure 16 shows the I/O Bandwidth (GB/s) and IOPS of our file system throughout our full system ImageNet with `PyTorch` training using 64 GPUs. Between 10th and 60th epoch, the average bandwidth is 3.84 GB/s and the average IOPS is 42.95K.

[33] https://github.com/NVIDIA/nccl
[34] https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks
[35] https://github.com/NVIDIA/apex
[36] https://github.com/richardkxu/distributed-pytorch

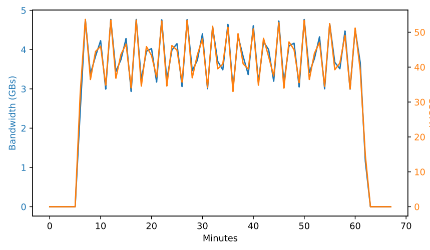

Figure 16: ImageNet ResNet-50 Full System Training I/O Bandwidth with 64 GPUs.

5 DISCUSSION AND LESSONS LEARNED

Since the initial deployment of the system in Spring 2019, we have issued over 300 user accounts, supporting over 70 faculty research groups from over 20 departments. The system provides computing cycles that otherwise would be cost-prohibitive for many research groups to acquire elsewhere. For example, the cost of a single V100 GPU node on AWS (p3.2xlarge instance) is \$3.06/hour[37]. Using this cost as a reference, HAL provides over \$141,000.00 in value every month.

Numerous papers have been published based on the work carried out on HAL. They range from visual scene analysis [8] to graph structured prediction energy networks [4] to gravitational wave denoising of binary black hole mergers [12]. The system also has been used by students for classwork and independent study projects. Undergraduate students working at NCSA on various research projects have been using the system since its introduction.

We have learned that storage selection plays a key role in making the HAL system usable. The high IOPS of our storage ensures good performance for all users running on the system, preventing demanding workloads from slowing the work of others or negatively impacting the interactive gateways of the system.

Another lesson learned is the need for flexibility with supporting various user requirements. `WMLCE` provides a set of specific versions of DL tools which cannot be easily upgraded. Therefore, we support containers via `Singularity`. The system admin team can build container images based on user requirements. We also support "fakeroot" feature to give the users administrative rights inside the container to enable them install 3rd party software. A "fakeroot" user cannot access or modify files and directories for which they do not already have access rights on the host filesystem [7].

Using HAL for executing complex multi-node DL training tasks is non-trivial even for advanced users. Therefore, we provide regular training sessions for new users covering topics ranging from how to get started using HAL to how to run distributed DL frameworks. We started with an intensive two-day training workshop in Spring 2019[38] and continue with short weekly training sessions in Fall 2019[39] and Spring 2020[40]. Training materials presented at these events are openly available. We also provide user support through `Jira` ticketing system, `Slack` channel, and walk-in consultations.

[37]https://aws.amazon.com/ec2/instance-types/p3
[38]http://www.ncsa.illinois.edu/enabling/data/deep_learning/news/powerai2019
[39]http://www.ncsa.illinois.edu/enabling/data/deep_learning/news/hal_fall19
[40]http://www.ncsa.illinois.edu/enabling/data/deep_learning/news/hal_spring20

To stimulate interest in ML/DL, we organize hackathons where students can work on various problems that require building and training DNN models. Two such hackathons have been organized in Fall 2019[41],[42] and one in Spring 2020[43].

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation's Major Research Instrumentation program, grant No. 1725729, as well as the University of Illinois at Urbana-Champaign. We would like to thank Daniel Lapine from NCSA, Steve Zehner, Trish Froeschle, Thomas Prokop, Jamie Syptak, and Terry Leatherland from IBM for their expert advise and invaluable contributions to the development and operation of the system. We would also like to thank our student intern Nishant Dash for his contribution to the development of `swsuite` software.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[2] Alexandre Bicas Caldeira. 2018. *IBM power system AC922 introduction and technical overview.* IBM Corporation, International Technical Support Organization.
[3] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).
[4] Colin Graber and Alexander Schwing. 2019. Graph Structured Prediction Energy Networks. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8690–8701. http://papers.nips.cc/paper/9074-graph-structured-prediction-energy-networks.pdf
[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.
[6] Dave Hudak, Doug Johnson, Alan Chalker, Jeremy Nicklas, Eric Franz, Trey Dockendorf, and Brian McMichael. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622. https://doi.org/10.21105/joss.00622
[7] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PloS one* 12, 5 (2017).
[8] J. Lin, U. Jain, and A. G. Schwing. 2019. TAB-VCR: Tags and Attributes based VCR Baselines. In *Conference on Neural Information Processing Systems (NeurIPS).*
[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems.* 8024–8035.
[10] Dino Quintero, Bing He, Bruno C Faria, Alfonso Jara, Chris Parsons, Shota Tsukamoto, Richard Wale, et al. 2019. *IBM PowerAI: Deep Learning Unleashed on IBM Power Systems Servers.* IBM Redbooks.
[11] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).
[12] Wei Wei and E.A. Huerta. 2020. Gravitational wave denoising of binary black hole mergers with deep learning. *Physics Letters B* 800 (2020), 135081. https://doi.org/10.1016/j.physletb.2019.135081
[13] Yang You, Zhao Zhang, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2017. Imagenet training in 24 minutes. *arXiv preprint arXiv:1709.05011* (2017).

[41]http://www.ncsa.illinois.edu/enabling/data/deep_learning/news/2019_ai_hack_1
[42]http://www.ncsa.illinois.edu/enabling/data/deep_learning/news/2019_ai_hack_2
[43]http://www.ncsa.illinois.edu/enabling/data/deep_learning/news/2020_ai_hack_3