Berzelius

25-10-2021

Trusted partner for your **Digital Journey**

Atos

Berzelius SuperPod

- Fastest Super Computer in Sweden – 300PetaFLOPS AI
- In the Top10 European Super computer
- In the Top20 of Nvidia Accelerated System WorldWide
- #89 on Top500 (5.253 TFLOPS HPL)

Berzelius SuperPOD Configuration

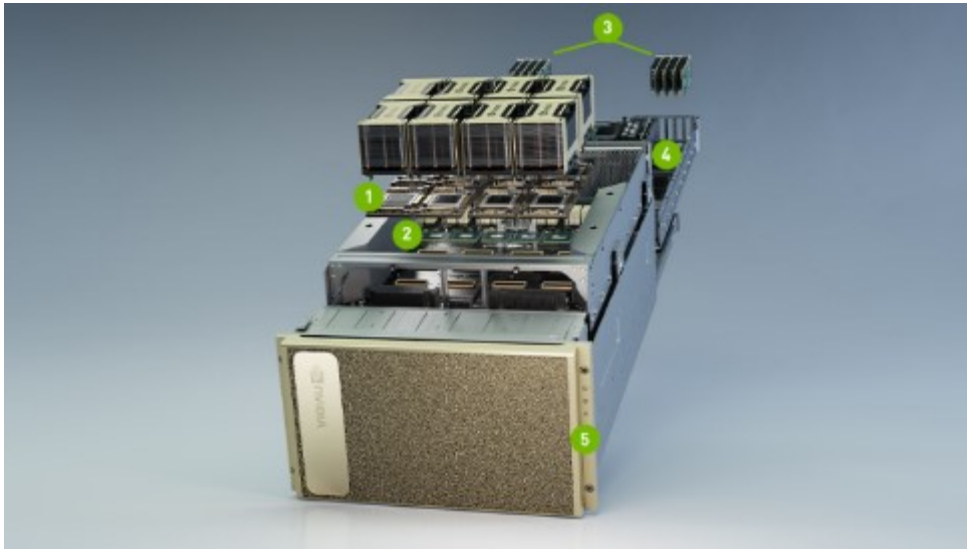
- 60 DGX A100
- Fully non Blocking Compute Fabric : 44x Nvidia HDR 200Gbs Switches
- Fully non Blocking :6x Nvidia HDR 200Gbs Switches
- Fast Parallel shared Storage : 4x AI400X
- Users Access : 3x Login Nodes
- Administrator Access : 2 Management Servers

Your SuperPOD Configuration

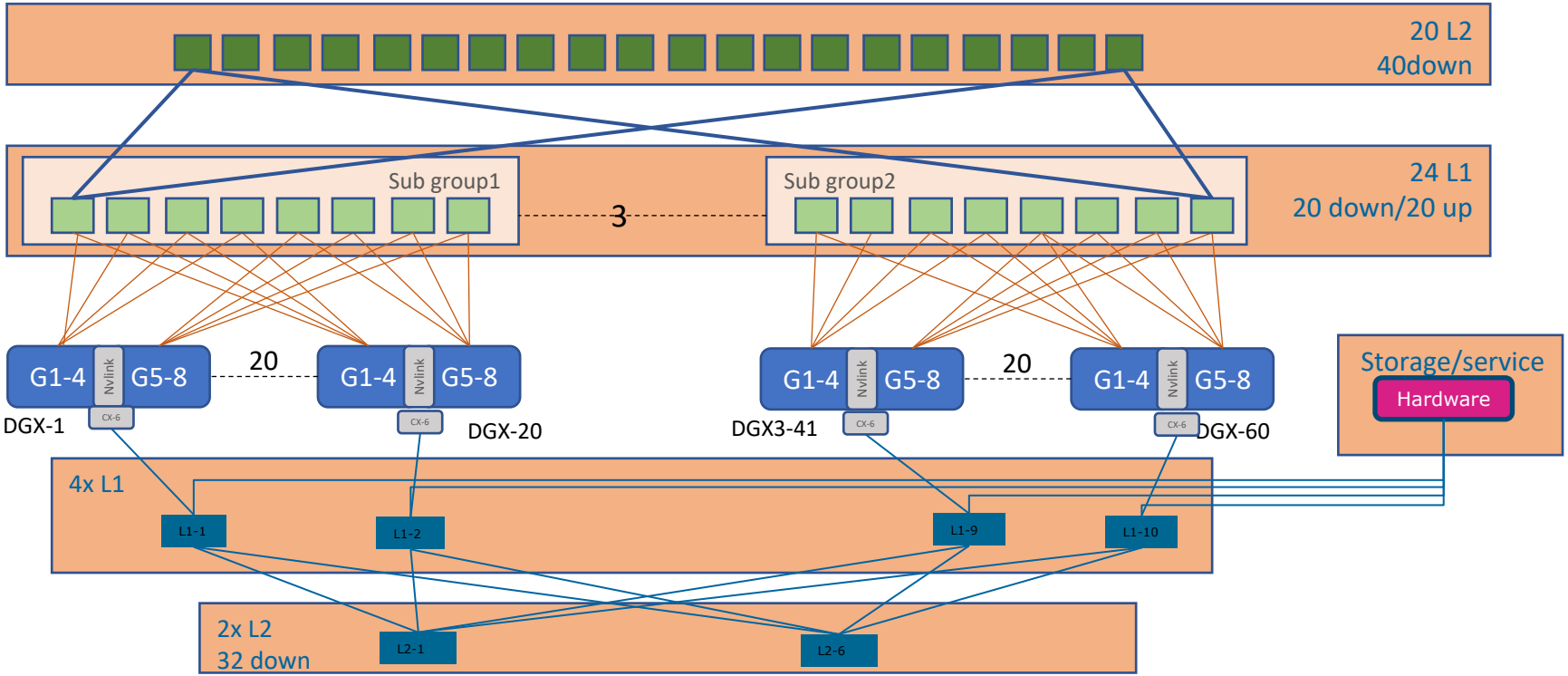
AI Compute Engine: Nvidia DGX-A100 80GB Server

Performance : 5 PFLOPS AI
Maximum Power Usage : 6.5kW

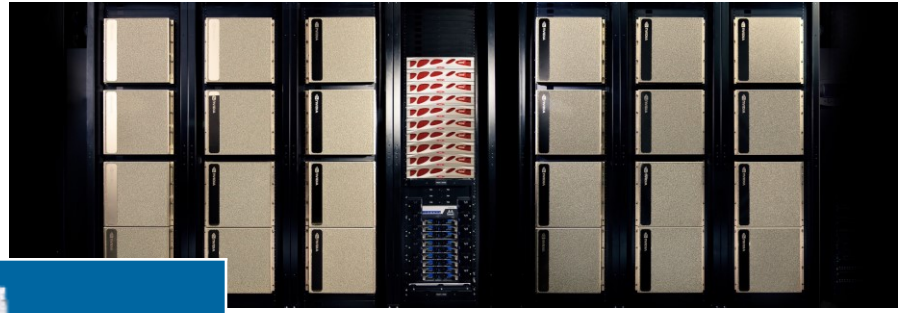
1. 8X NVIDIA A100 GPUS WITH 320 GB TOTAL GPU MEMORY
   12 NVLinks/GPU, 600 GB/s GPU-to-GPU Bi-directonal Bandwidth
2. 6X NVIDIA NVSWITCHES
   4.8 TB/s Bi-directional Bandwidth, 2X More than Previous Generation NVSwitch
3. 9x MELLANOX CONNECTX-6 200Gb/S NETWORK INTERFACE
   500 GB/s Peak Bi-directional Bandwidth
4. DUAL 64-CORE AMD ROME7742, 2.25 GHz (base), 3.4 GHz (max boost) AND 1 TB SYSTEM MEMORY
   3.2X More Cores to Power the Most Intensive AI Jobs
5. 15 TB GEN4 NVME SSD AND 2x 1.92TB M.2 NVME FOR OS.
   25GB/s Peak Bandwidth, 2X Faster than Gen3 NVME SSDs



60x DGX SuperPod 1:1 Fabrics Using QM8700 HDR Switches



Storage
DDN AI400X Appliance

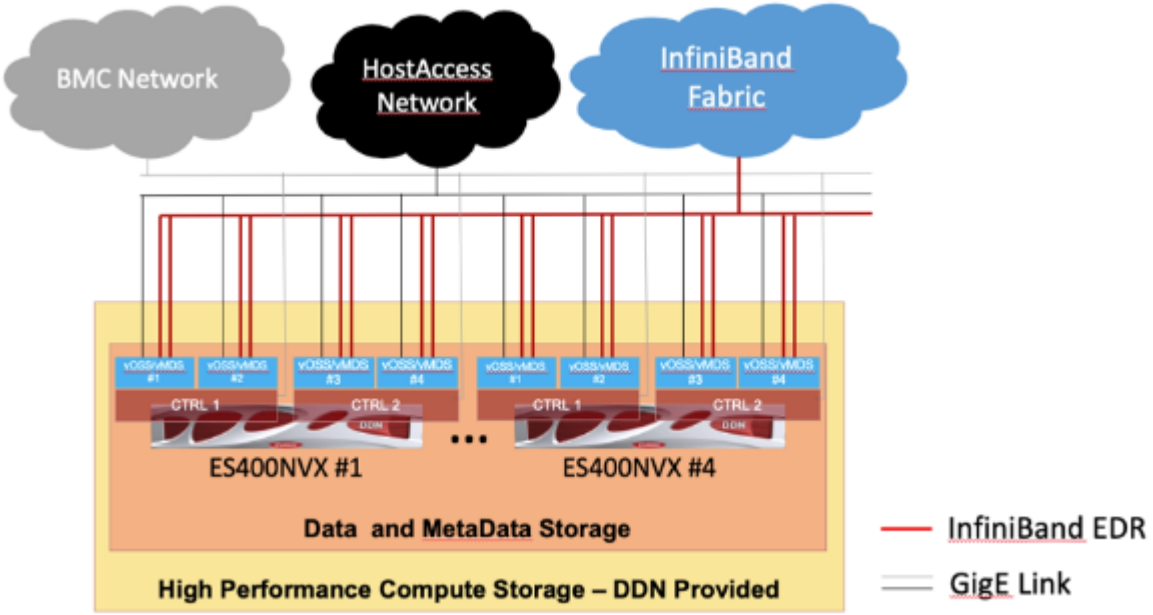


AI400NVXE	
CPU/Memory	4 Cascade Lake CPUs (Purley) 24 memory channels
Major Features	Dual port NVMe, tuned for IOPs, DeClustered RAID
Performance Targets	48 GB/s (read) 32 GB/s (write) > 3 Million IOP/s
Uplinks	IB: EDR/HDR100 (8), 40/100GbE (8)
Chassis Drive Slots	2U chassis : 24x 2.5” slots (Dual Port
SAS Downlinks	8x Mini-SAS HD ports (each is 4 lanes of 12Gb SAS) Support for 90 bay expansion (Add 0, 1, 2 or 4 SS9012)
Misc	PSU: 2x (N+1) 2000W Nominal, 80Plus Platinum rating Controller canister based Cache BBU

- Berzelius configuration
- 4x AI400NVX
  - ~192 GB/s Read Performance
  - ~128 GB/s Write Performance
  - ~1.4 PB raw capacity

Storage Scalability

Based on DDN Lutre Exa5 solution
Each Appliance runs 4x VM on each controlers
Each VM providing acces to Data through vOSS (virtual Object Storage Server) and to Metadata through vMDS (virtual MetaData Servers)



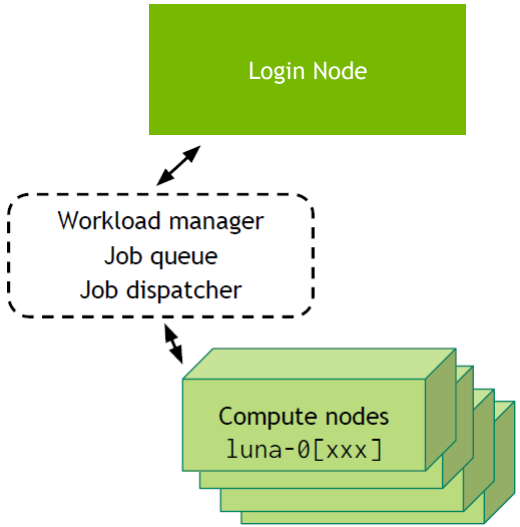
SLURM

“Simple Linux Utility for Resource Management”

There are **login nodes and compute nodes** on Berzelius.

When a user logs in, they will be on a login node. Login nodes don’t have GPUs and cannot run containers.

Users request compute nodes using via the workload manager. Our workload manager is Slurm.



All jobs executables and debugging sessions, should be
run on the compute nodes, berzelius[xxx].

SLURM

Basic usage

```
# running bare metal applicaiton
$ srun --ntasks 2 -l hostname
1: luna1
2: luna2
```

sbatch/srun Option	Description
-N, --nodes=	Specify the total number of nodes to request
-n, --ntasks=	Specify the total number of tasks to request
--ntasks-per-node=	Specify the number of tasks per node
-G, --gpus=	Total number of GPUs to allocate for the job
--gpus-per-task=	Number of gpus per task
--exclusive	Guarantee that nodes are not shared amongst jobs
-l	Show the line numbers
-p	Specify the name of the partition, here, please use the name of your tenant

SLURM BASIC

Viewing states

```
# Show the current running jobs
$ squeue
Tue Nov 17 19:08:18 2020
JOBID PARTITION NAME USER STATE TIME TIME_LIMIT NODES NODELIST(REASON)
9      batch    bash user01 RUNNING 5:43 UNLIMITED 1 dgx1
10     batch    Bash user02 RUNNING 6:33 UNLIMITED 2 dgx[2-3]
```

```
# show job info
$ scontrol show job 10
[exhaustive print out]

# Cancel a running job, using jobid 10
$ scancel 10
```

Maximum Job runtime is 12 hours!!!

SLURM CONTAINER STACK

Enroot and Pyxis

We still use Docker images. You can still use images from NGC.

BUT the Docker container runtime is not installed on Cambridge-1. You must use Pyxis to run containers instead. You can still run bare-metal applications too – although you will need to prepare the environment yourselves (installing the frameworks into you home directory).

```
# DOCKER workflow
#   (1) request a shell on a compute node
srun --pty bash
#   (2) start a shell inside a container
docker run --rm -it centos:7 bash

# now, you can run commands inside the container
cat /etc/os-release
```

```
# PYXIS workflow
#   (1) request a shell inside a container on a
compute node
srun --container-image=centos:7 --pty bash

# now, you are already inside the container; you
skipped the bare metal step
cat /etc/os-release
```

SLURM EXAMPLE 1

Interactive single-node job

```
# You can start a container by importing an image from NGC
$ srun –p tenant –G 8 –N 1 --container-name=pytorch --container-image=nvcr.io/nvidia/pytorch:21.03-py3 hostname

# We can reuse the existing container and execute our new binary on the dataset
$ srun –p tenant –G 8 –N 1 --container-name=pytorch --container-mounts=/mnt/datasets:/datasets vmtouch /datasets

# Now we can start an interactive session inside the container, for development/debugging:
$ srun –p tenant –G 8 –N 1 --container-name=pytorch --container-mounts=/mnt/datasets:/datasets --pty bash
root@superpod-01:/workspace#
```

SLURM EXAMPLE 2

Batch jobs

```
$ cat job.sh
#!/bin/bash
#!/bin/bash
#SBATCH -N 2
#SBATCH --gpus 16
#SBATCH --ntasks-per-node 8
#SBATCH --exclusive
#SBATCH -A berzelius-2021-43
#SBATCH --time=3:00:00

srun --ntasks=2 --mpi=pmix --container-image="nvcr.io#nvidia/tensorflow:20.12-tf2-py3" \
 --container-mount-home \
 /bin/bash -c "python /root/benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --num_gpus=1 --batch_size=200 --
model=resnet50 --variable_update=horovod --use_fp16"

$ sbatch --nodes=1 job.sh
```

CONTAINERS AND MULTI-NODE

slurm.schedmd.com/mpi_guide.html

```
# TensorFlow (and other MPI apps): no need to use mpirun for a multi-node launch
srun -p tenant -G 128 --nodes=16 --ntasks-per-node=8 --mpi=pmix --container-image=tensorflow/tensorflow \
    --container-mounts=/raid/datasets:/datasets \
    python train.py /datasets/dogs
# PyTorch: no need to use PMIx for a multi-node launch, --mpi=none is optional
srun -p tenant -G 128 --nodes=16 --ntasks-per-node=8 --mpi=none --container-image=pytorch/pytorch \
    --container-mounts=/raid/datasets:/datasets \
    python train.py /datasets/cats
```

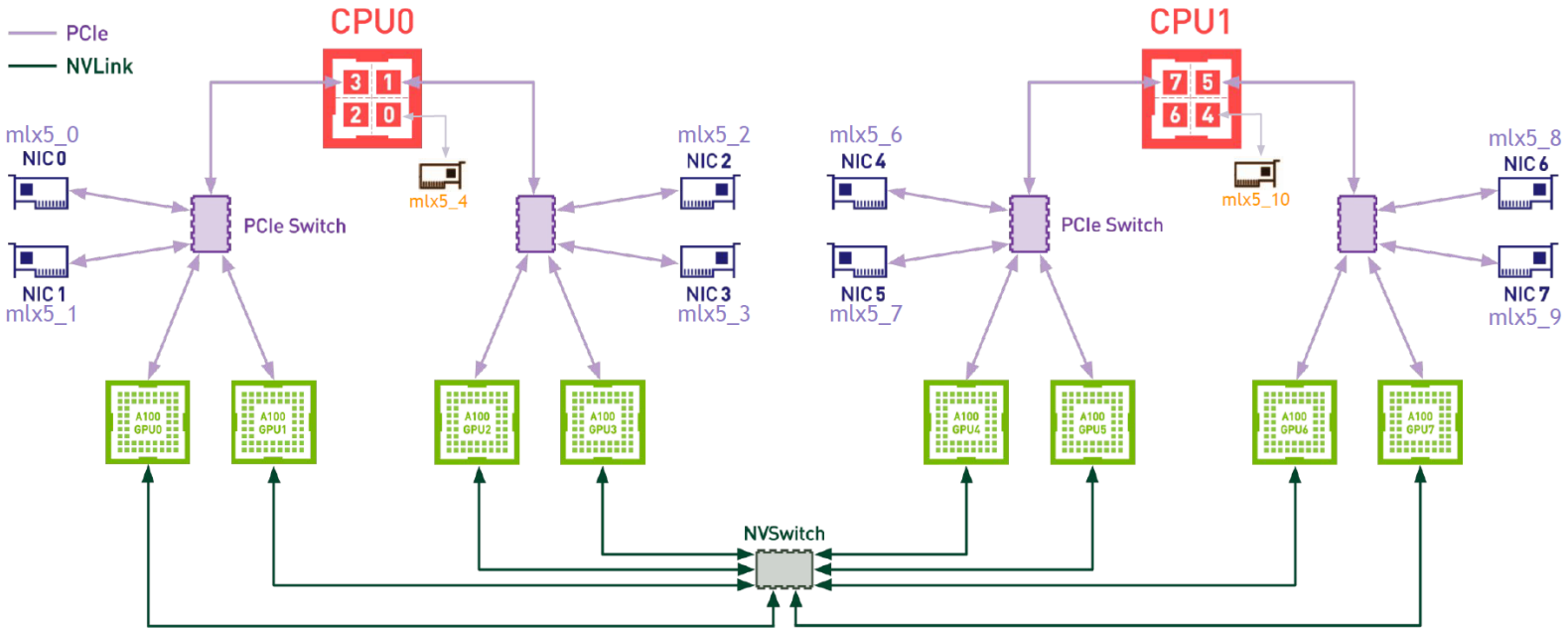
TIP: MAKE SLURM OUTPUT MORE READABLE

Use the -l option when using srun

```
# TensorFlow (and other MPI apps): no need to use mpirun for a multi-node launch

srun -l -p tenant -G 128 --nodes=16 --ntasks-per-node=8 --mpi=pmix --container-image=tensorflow/tensorflow \

     --container-mounts=/raid/datasets:/datasets \

     python train.py /datasets/dogs



1: importing Pyxis Container
8: importing Pyxis container
[…]
```

WHAT IS DGX A100 80GB?

High-level Topology Overview (for performance tuning purposes)



PERFORMANCE: BINDINGS

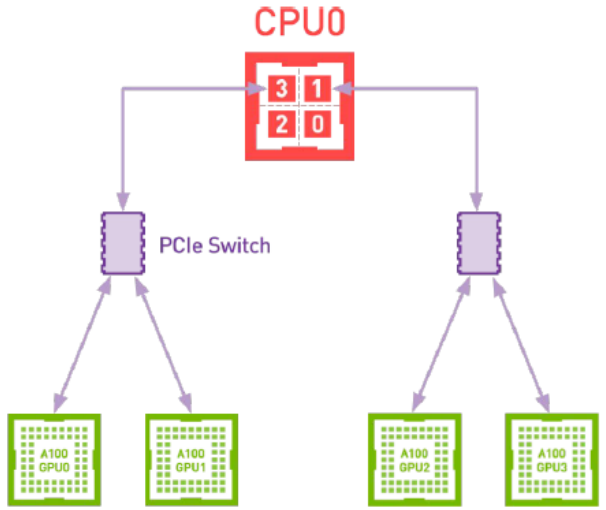
Half the NUMA nodes do not have a GPU affine to it.

If you rely on nvmlDeviceGetCpuAffinity for bindings, you will be using half the available cores/memory bandwidth.

This might be fine if your app is not CPU/memory intensive.

DALI has a flag to disable binding for the nvJPEG threads: ImageDecoder(affine=false)

Not all applications play nicely with 256 hw threads on a system. They might crash/slowdown with too many worker threads (e.g. 1 per core).



PERFORMANCE: BINDINGS

Basic recommendations

Recommendations for basic NUMA bindings:

- Test with socket-level affinity:
  - GPUs [0,1,2,3]: numactl --cpunodebind=0-3 --membind=0-3
  - GPUs [4,5,6,7]: numactl --cpunodebind=4-7 --membind=4-7
- Test with the affinity reported by nvidia-smi topo -m:
  - GPUs [0,1]: numactl --cpunodebind=3 --membind=3
  - GPUs [2,3]: numactl --cpunodebind=1 --membind=1
  - GPUs [4,5]: numactl --cpunodebind=7 --membind=7
  - GPUs [6,7]: numactl --cpunodebind=5 --membind=5
- Keep the best of the two bindings.