Containerizing Deep Learning Frameworks with Singularity

Rengan Xu, Frank Han, Nishanth Dandapanthula

HPC & AI Solutions Engineering, Dell EMC

Agenda

- Dell EMC HPC & AI Solutions Engineering
- Why use containers?
- Singularity Containers
  - Singularity vs Docker
  - Interpretability between Singularity vs Docker
  - Singularity workflow
- Containerizing DL frameworks
  - Issues and workarounds
  - eg. Caffe2
- Performance Results
  - Horovod + TensorFlow
  - MXNet
  - Caffe2

Dell EMC HPC & AI Solutions Engineering

Design, develop and integrate HPC systems

Conduct application performance studies and develop best practices

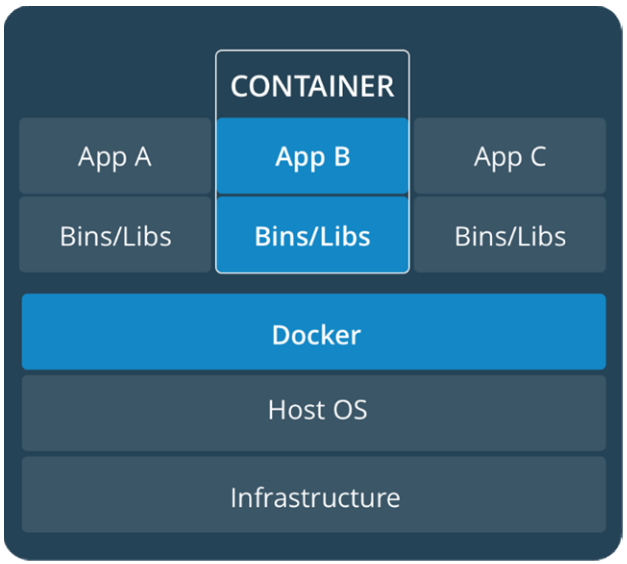
HPC & AI Innovation Lab

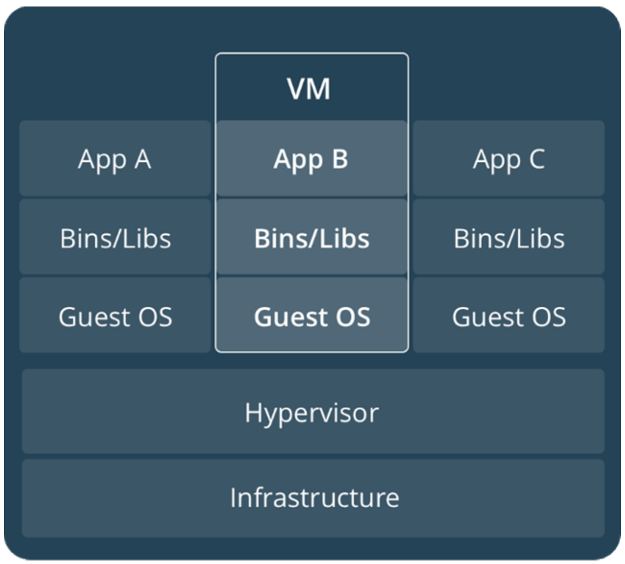
Act as the focal point for joint R&D activities

Prototype and evaluate advanced technologies

Containers and Virtualization Machine: A Recap

- Container has no hypervisor
- Container has no guest OS





source: https://www.docker.com/what-container

DELL EMC

Need for Containerization

- Why do we need containers?
  - Simplify application building
  - Application isolation
  - Faster application deployment
  - Validate and reproduce results
  - Server consolidation/Server efficiency
  - Can be deployed on bare metal or on virtual machines

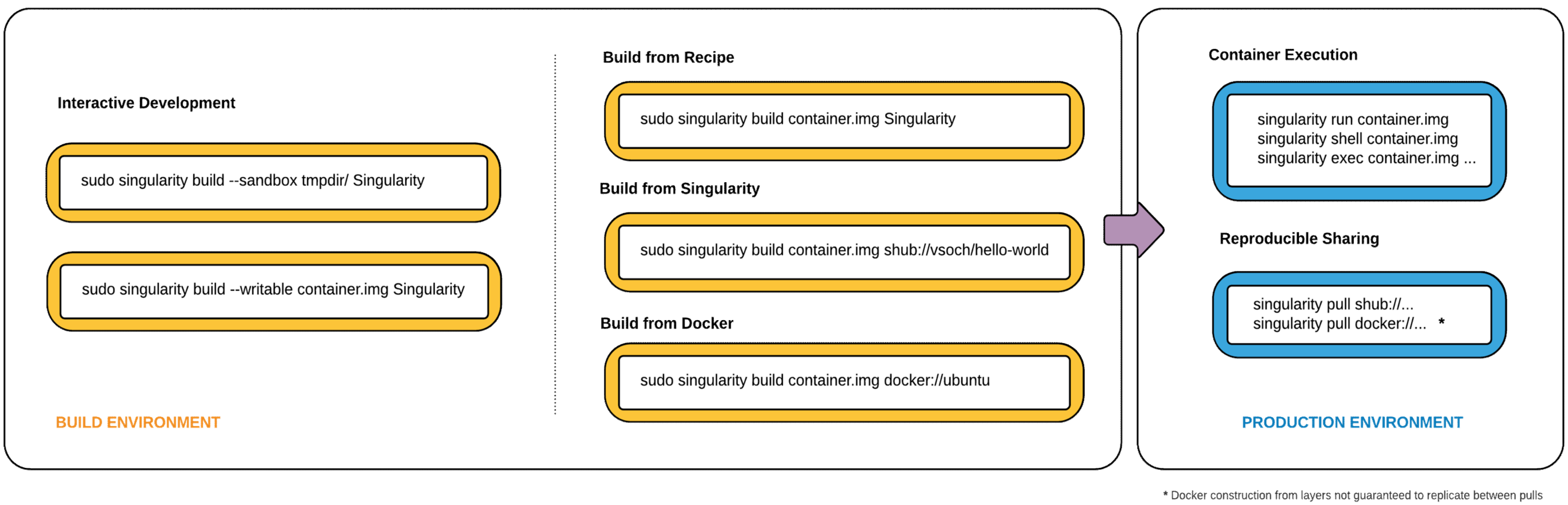
- Benefits of Containers
  - Lightweight
  - Low overhead
  - Easier application sharing among users
  - Reproducibility

- Example containers
  - LXC
  - Docker
  - Singularity

Singularity Vs Docker

Feature	Singularity	Docker
Multiple containers can be run on same hardware	✓	✓
Can be created and destroyed more quickly	✓	✓
Do not need entire OS, only a core run time	✓	✓
Transferable to other machines easily	✓	✓
Image format	Single file	Layered Image
Use with HPC schedulers	✓	X
Native Support for MPI	✓	X
Support for GPUs	✓	X
root owned Daemon process	X	✓

Singularity: Workflow Summary



source: http://singularity.lbl.gov/docs-flow

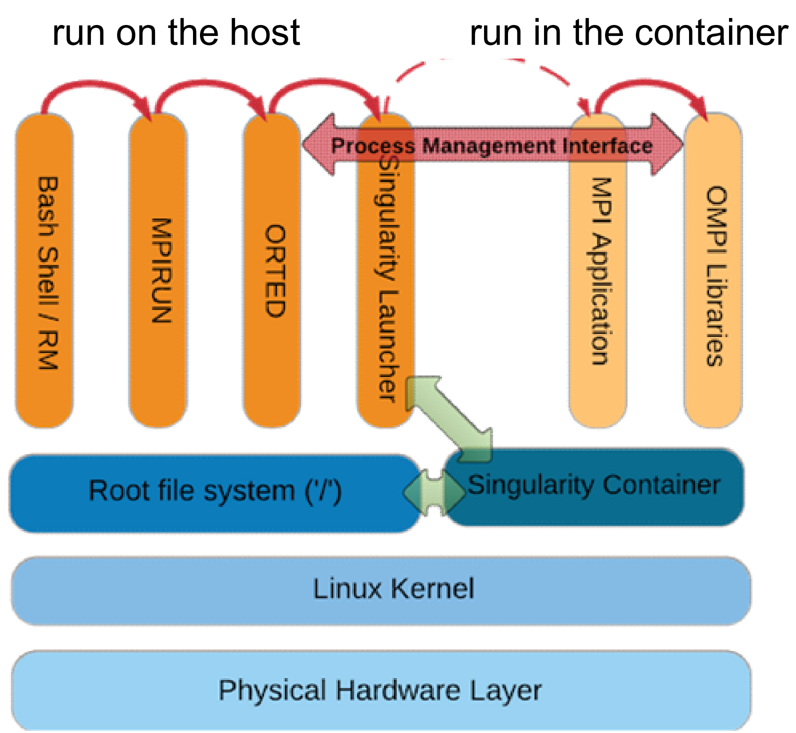
DELL EMC

Interpretability between Singularity vs Docker

- Create Singularity image from Docker Hub
  - $ singularity pull docker://tensorflow/tensorflow
- Create Singularity image from Nvidia GPU Cloud Docker Registry
  - $ export SREGISTRY_NVIDIA_BASE="ngcr.io“
  - $ export SREGISTRY_CLIENT=nvidia
  - $ export SREGISTRY_NVIDIA_USERNAME='$oauthtoken‘
  - $ export SREGISTRY_NVIDIA_TOKEN='[NGC_API_KEY]‘
  - $ sregistry pull nvidia://tensorflow:17.11

Singularity MPI

- Has built-in support for all MPI implementations (OpenMPI, MPICH, Intel MPI, etc.)
- Host MPI version must be newer or equal to the version inside the container
- Example:
  - mpirun –np 4 **singularity** exec centos_ompi.img /usr/bin/mpi_ring



source: https://wikihub.berkeley.edu/download/attachments/129695919/Containers_in_HPC_summary_Singularity.pdf

Challenges and Workarounds

- Why containerize DL Frameworks
  - Every DL framework has too many dependences
  - Each dependent library has special version requirement
  - All DL frameworks are changing frequently
  - The friendly supported OS for most DL frameworks is Ubuntu, where as datacenter deployments are RHEL/Centos
- Why we moved to singularity
  - Scaling containerized deep learning frameworks past a single node
- Issues faced with Singularity
  - PCIe device driver mismatch
- Workarounds
  - GPUs
    - The container should always use the host GPU driver
    - Create a symbolic links for all GPU driver related files and then bind it to container
    - Update to latest drivers since they are backward compatible
  - InfiniBand
    - The InfiniBand driver is kernel dependent, and the solution is to make the container OS and host OS compatible and the container reuses the InfiniBand driver and libraries on the host

Singularity recipe for Caffe2

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum

%post
    yum -y groupinstall "Development Tools"
    yum -y install epel-release which vim time wget yum-utils
    yum -y install scipy python-devel python-pip protobuf-devel glog-devel gflags-devel hdf5-devel
    yum -y install opencv-devel opencv-python lmdb-devel leveldb-devel snappy-devel atlas-devel
    yum -y install openblas-devel hiredis-devel eigen3-devel libcurl-devel bc
    pip install flask future graphviz hypothesis jupyter matplotlib numpy protobuf pydot python-nvd3
    pip install pyyaml requests scikit-image scipy setuptools six tornado

    wget https://cmake.org/files/v3.10/cmake-3.10.2.tar.gz
    tar xzfv cmake-3.10.2.tar.gz
    cd cmake-3.10.2
    ./bootstrap --system-curl
    make -j8
    make install

%environment
    LANG=en_US.UTF-8
    LANGUAGE=en_US:en
    LC_ALL=en_US.UTF-8
    export LANG LANGUAGE LC_ALL
```

Building the container

```
#!/bin/bash

source ./caffe2.config

echo "Type the paths (separated by ':') to bind from host OS:"
read bind_paths

module load shared singularity/2.4.2
singularity build --sandbox centos7_caffe2_dev_sandbox centos7.def

#bind_paths="/cm/shared:/cm/local"
to_bind_paths=$(echo $bind_paths | tr ":" "\n")
bind_flags=""
for path in $to_bind_paths
do
    singularity exec -s /bin/bash -w centos7_caffe2_dev_sandbox mkdir -p $path
    bind_flags="$bind_flags -B $path"
done

ln -s $GPU_DRIVER_PATH/libcuda.so* $PWD/host_libs/
ln -s $GPU_DRIVER_PATH/libnvidia* $PWD/host_libs/
ln -s $CUDNN_PATH $PWD/host_libs/cudnn
ln -s $NCCL_PATH $PWD/host_libs/nccl
ln -s /usr/bin/nvidia-smi $PWD/host_libs/nvidia-smi

singularity exec -s /bin/bash -w centos7_caffe2_dev_sandbox mkdir -p /etc/libibverbs.d
# /ibverb_libs will bind the host libibverbs.so path (default path is /usr/lib64 on host)
# The reason that the container does not bind to /usr/lib64 is to avoid files conflict in container
singularity exec -s /bin/bash -w centos7_caffe2_dev_sandbox mkdir -p /ibverb_libs

singularity exec -s /bin/bash $bind_flags -B $PWD:/mnt -w centos7_caffe2_dev_sandbox /mnt/build_caffe2.sh
```


DELL EMC

Build Caffe2 inside the container

```bash
#!/bin/bash
set -x
source /mnt/caffe2.config

export INCLUDEPATH=$CUDA_PATH/include:$MPI_PATH/include:/mnt/host_libs/cudnn/include:/mnt/host_libs/nccl/include:$INCLUDEPATH
export C_INCLUDE_PATH=$CUDA_PATH/include:$MPI_PATH/include:/mnt/host_libs/cudnn/include:/mnt/host_libs/nccl/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$CUDA_PATH/include:$MPI_PATH/include:/mnt/host_libs/cudnn/include:/mnt/host_libs/nccl/include:$CPLUS_INCLUDE_PATH
export LD_LIBRARY_PATH=$CUDA_PATH/lib64:$MPI_PATH/lib:/mnt/host_libs:/mnt/host_libs/cudnn/lib64:/mnt/host_libs/nccl/lib:$LD_LIBRARY_PATH
export LIBRARY_PATH=$CUDA_PATH/lib64:$MPI_PATH/lib:/mnt/host_libs:/mnt/host_libs/cudnn/lib64:/mnt/host_libs/nccl/lib:$LIBRARY_PATH

export PATH=$MPI_PATH/bin:$CUDA_PATH/bin:/mnt/host_libs:$PATH

cd /mnt
git clone --recursive https://github.com/caffe2/caffe2

cd caffe2 && mkdir -p build_openmpi
cd build_openmpi

cmake .. -DUSE_CUDA=ON -DUSE_REDIS=ON -DUSE_GLOO=ON -DUSE_NNPACK=ON -USE_MPI=ON \
    -DCUDNN_INCLUDE_DIR=/mnt/host_libs/cudnn/include \
    -DCUDNN_LIBRARY=/mnt/host_libs/cudnn/lib64/libcudnn.so \
    -DNCCL_INCLUDE_DIR=/mnt/host_libs/nccl/include \
    -DNCCL_LIBRARIES=/mnt/host_libs/nccl/lib/libnccl.so \
    -DHiredis_INCLUDE_DIR=/usr/include/hiredis \
    -DHiredis_LIBRARIES=/usr/lib64/libhiredis.so | tee config.log
make -j16 VERBOSE=1 | tee compile.log
```

Run the container

- ${mpirun_options} ${profile_options} \

  singularity exec -s /bin/bash \

  -B $host_paths -B $PWD:/mnt \

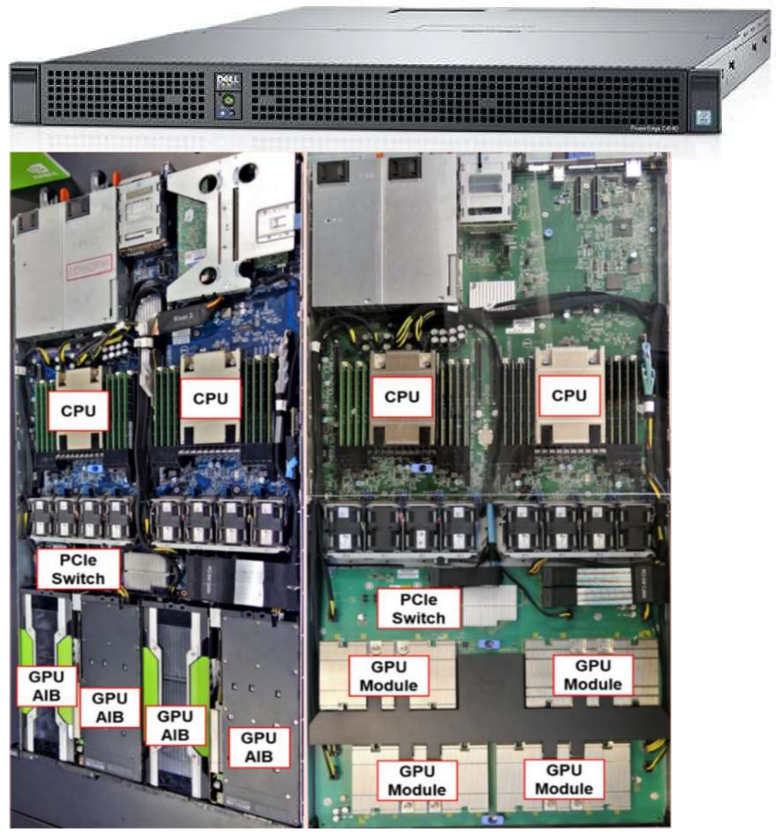
  -B /usr/lib64:/ibverb_libs -B /etc/libibverbs.d -B /sys/class/infiniband_verbs \

  centos7_caffe2_dev_sandbox /mnt/caffe2_singularity_cmd.sh \

  ${WORK_DIR} ${gpu_arch} ${gpus_per_node} $network ${run_id} ${num_nodes} $epochs $profile $debug $mpi) >& $train_log

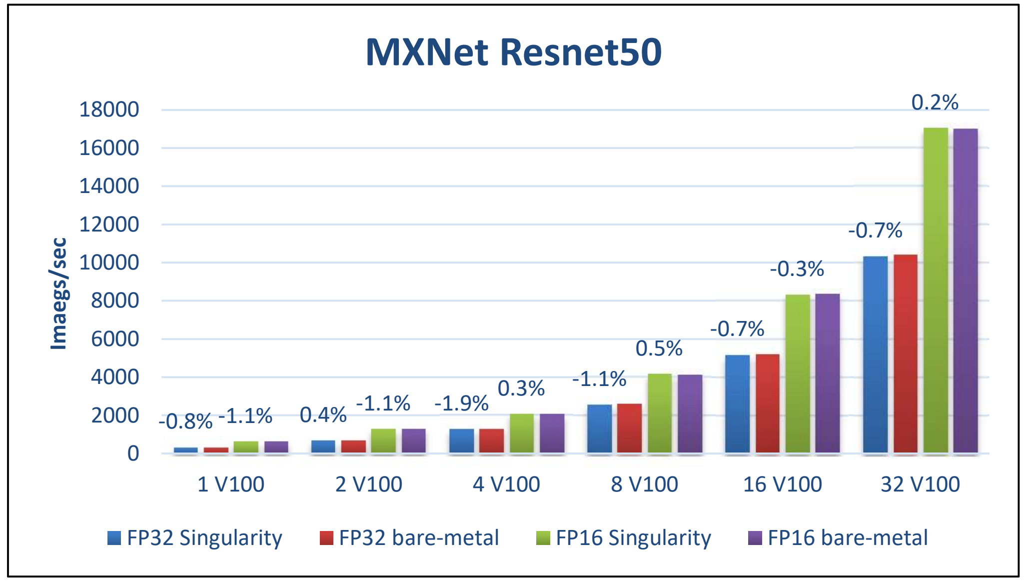
Testbed

- 8 Dell EMC PowerEdge C4140 nodes.
  - In process of updating to 32 nodes with NVLINK
- Nvidia V100-PCIe GPUs
- Intel Xeon Skylake CPU
- Mellanox 100Gbps EDR Infiniband
- CUDA 9.0, CUDNN 7.0, NCCL 2.0
- Dataset: ILSVRC 2012



Performance Results – MXNet

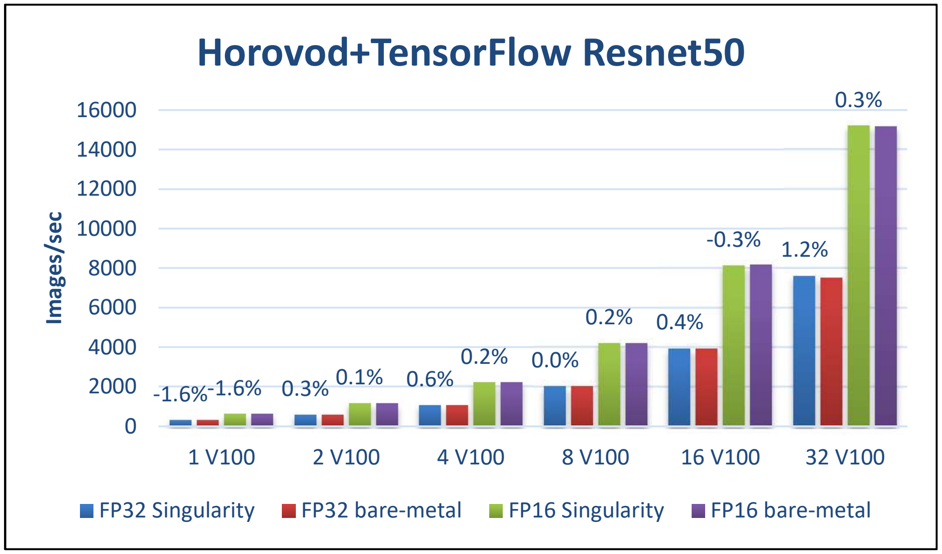
- In FP32 mode, batch size: 64 per GPU
- In FP16 mode, batch size: 128 per GPU
- IPoIB, rsync are used for nodes communication
- Speedup of 32 V100 is 29.4x in FP32 and 25.8x in FP16



Performance difference between Singularity vs bare-metal

Performance Results – Horovod + TensorFlow

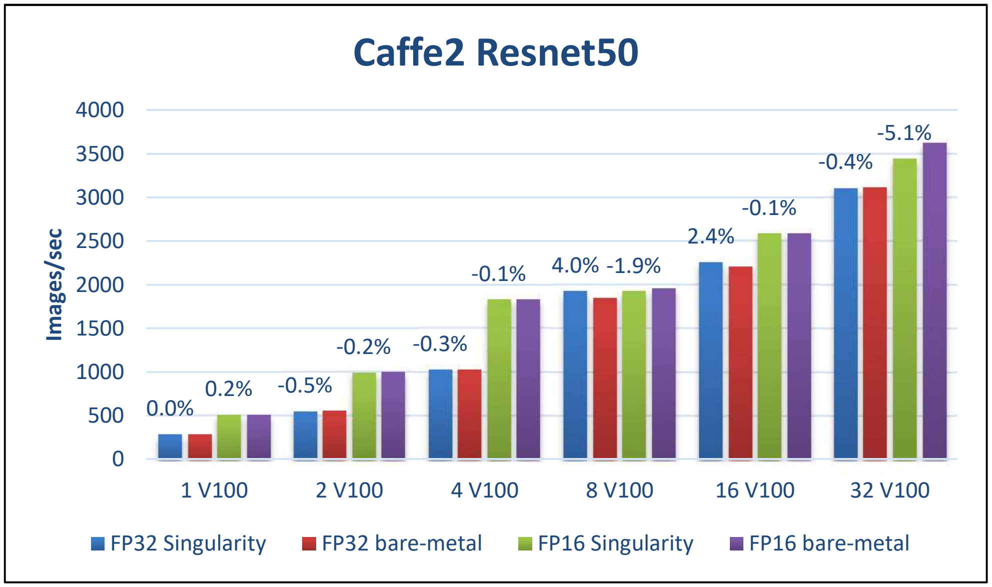
- In FP32 mode, batch size: 128 per GPU
- In FP16 mode, batch size: 256 per GPU
- MPI used for multi-node communication
- Speedup of 32 V100 is 22.4x in FP32 and 23.7x in FP16



Performance difference between Singularity vs bare-metal

Performance Results – Caffe2

- In FP32 mode, batch size: 64 per GPU
- In FP16 mode, batch size: 128 per GPU
- Redis and IPoIB are used for nodes communication
- Caffe2 performance unstable on multiple nodes



Performance difference between Singularity vs bare-metal

Conclusions and Future Work

- Conclusions
  - Singularity simplifies the building and deployment of DL in both single-node and multi-node
  - Easy to use Singularity on GPU server
  - Straightforward to run MPI on InfiniBand interconnect
  - No performance loss compared to bare-metal
- Future Work
  - File system impact for DL models
  - Scale impact for DL model accuracy
  - Research on neural networks with model parallelism
  - Case studies with appropriate DL models
- Build Optimal Solutions targeted to DL vertical.

www.hpcatdell.com

{**Rengan.Xu**,Frank.Han1,Nishanth.Dandapanthu}@Dell.com

DELL EMC