Leveraging HPC to Accelerate Machine Learning

Weiguang Guan
SHARCNet/CC
guanw@sharcnet.ca

Goal

✔️ To learn how to efficiently train DNN on HPC clusters

- Data I/O
- CPUs vs GPUs
- Multiple GPUs on a single node
- Distributed multi-GPU across multiple nodes

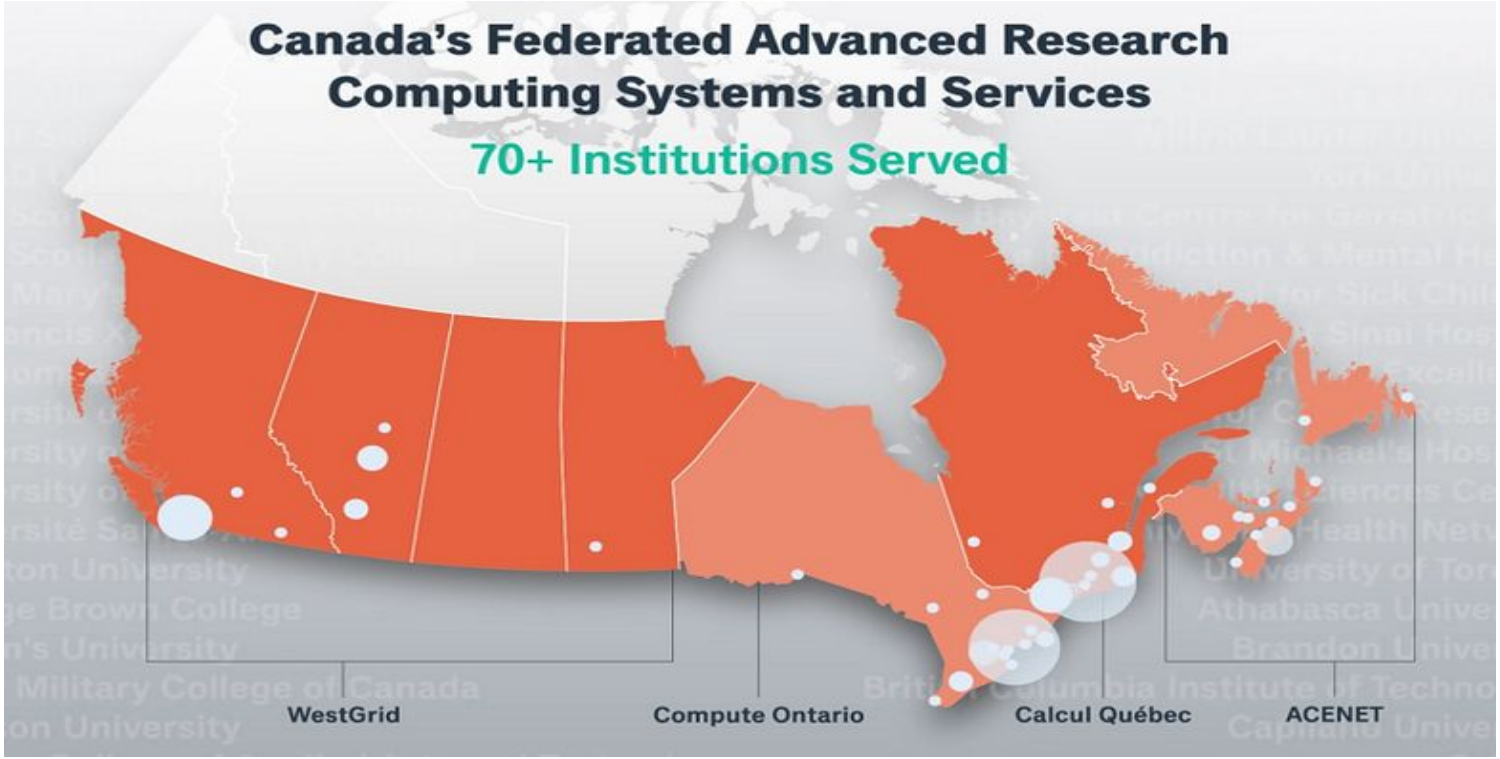
❌ Not a tutorial on

- Machine learning

Outline

- Introduction to HPC clusters in Canada
- Overview of distributed parallel NN training
- Regular NN training ⇨ distributed NN training on a single node
  - Through different APIs: Keras, Estimator
  - Performance benchmarks of using single/multiple CPUs/GPUs
- Distributed NN training across cluster (HOROVOD)

Compute Canada (CC)



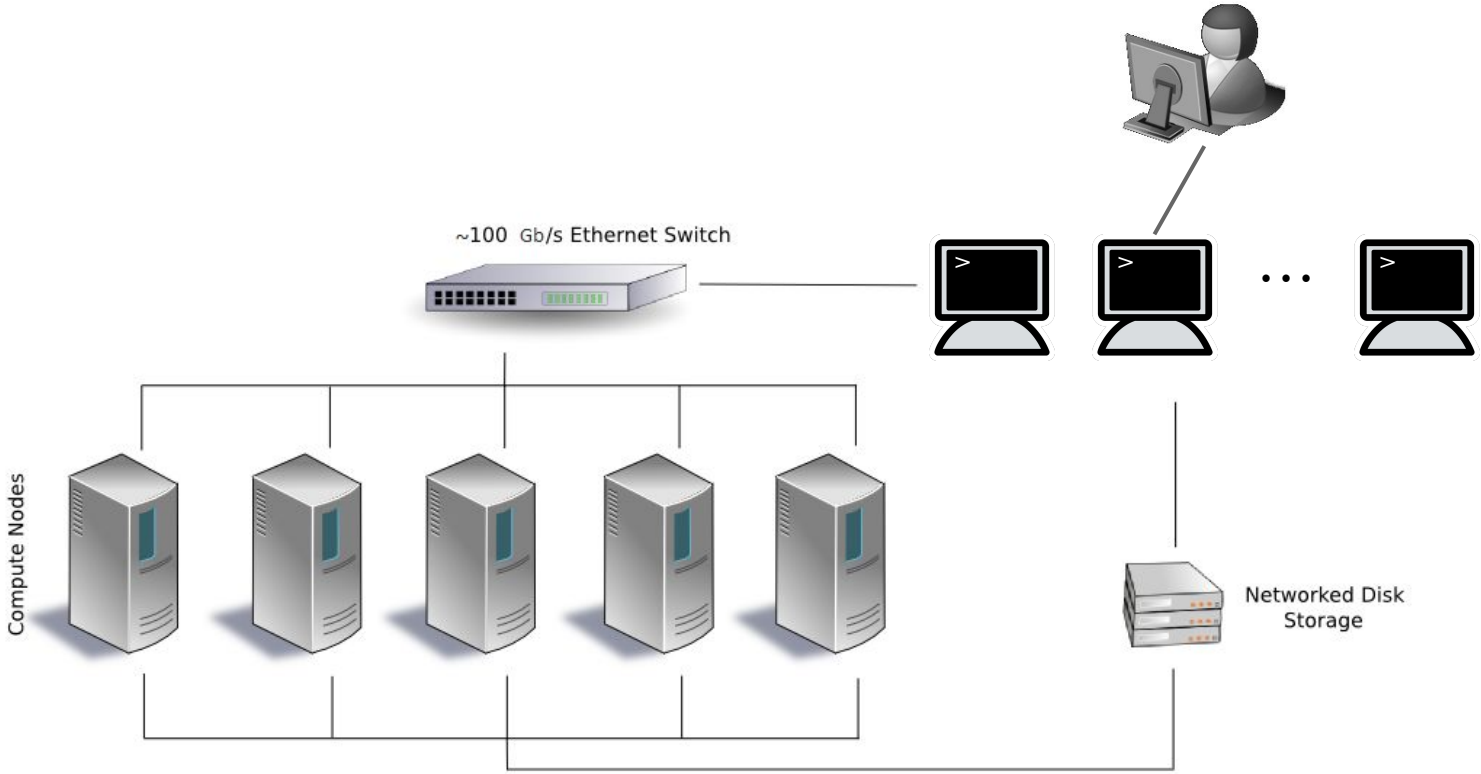
Resources of Compute Canada

- Beluga (Calcul Quebec)
- Graham (SHARCNet)
- Cedar (Westgrid)
- Niagara (SciNet)
- Clouds
  - East cloud
  - Arbutus
  - Graham cloud

Graham cluster operated by SHARCNet

- 884 base nodes (32 cpu cores, 125GB memory)
- 56 nodes (32 cpu cores, 250GB memory)
- 24 nodes (32 cpu cores, 502GB memory)
- 3 nodes (64 cpu cores, 3022GB memory)
- 160 GPU nodes (32 cpu cores, 124GB memory, 2 nVidia Pascal p100 GPUs)
- 7 AI GPU nodes (28 cpu cores, 178GB memory, 8 nVidia Volta v100 GPUs)

Architecture of cluster



Computing environment

- **Operating systems**: Linux (64-bit CentOS)
- **Languages**: C/C++, Fortran, Matlab/Octave, Python, R, Java, etc.
- **Parallel development support**: MPI, pthreads, OpenMP, CUDA, OpenACC, OpenCL
- **Software modules**: select pre-built and configured software, as well as versions, with the module commands
- **Job Scheduling**: SLURM scheduler

File system	Quotas	Backed up?	Purged?	Available by Default?	Mounted on Compute Nodes?
Home Space /home	50 GB/0.5M files per user	Yes	No	Yes	Yes
Scratch Space /scratch	20 TB/1M files per user. Extendable to 100 TB	No	Yes, all files older than 60 days	Yes	Yes
Project Space /project	1 TB and 0.5M files per group, can request increase to 10 TB	Yes	No	Yes	Yes
Local disk	>960GB SSD	No	No	Transient	Per node

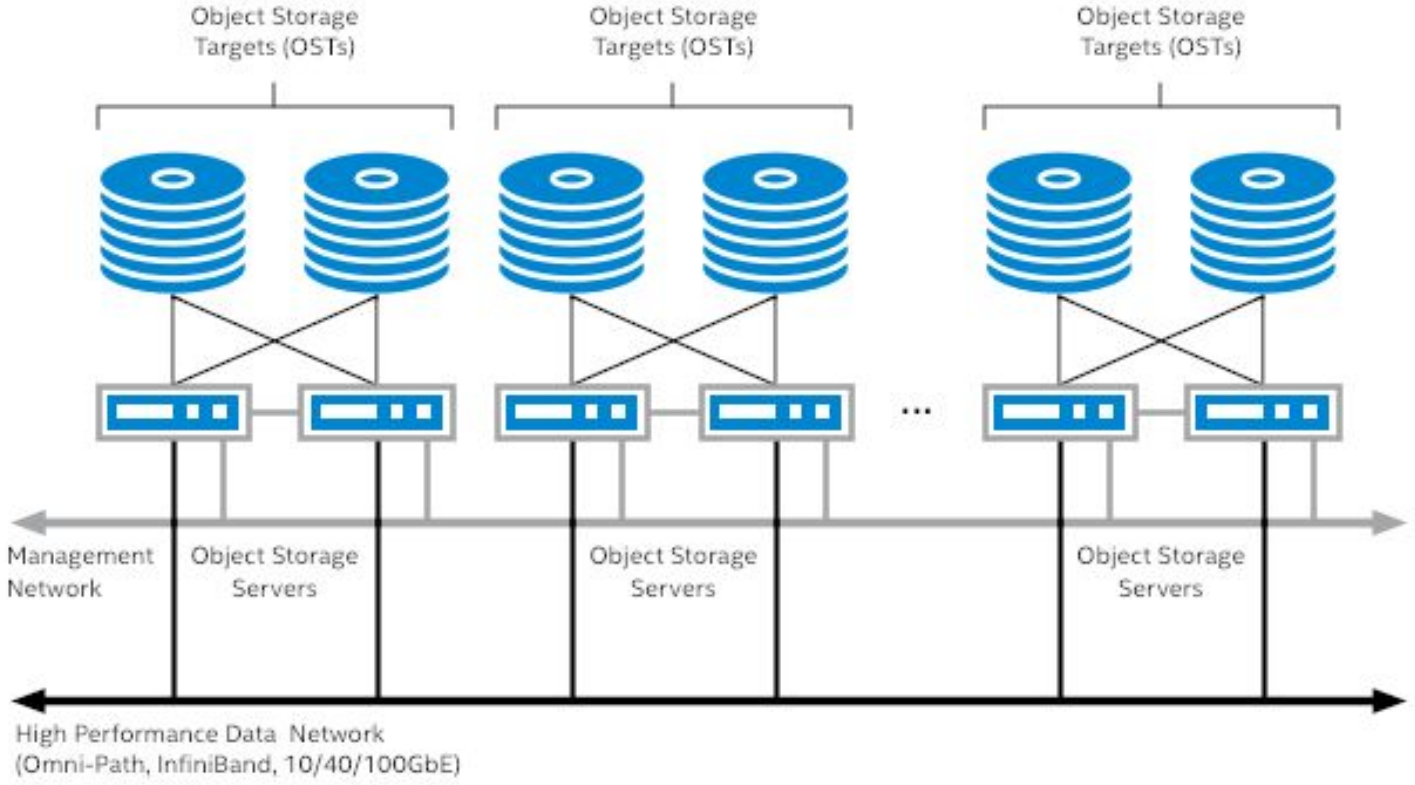
File systems

- /home (nfs)
- /project (lustre)
- /scratch (lustre)
- Fast local disk: $SLURM_TMPDIR (SSD or RAMDisk)

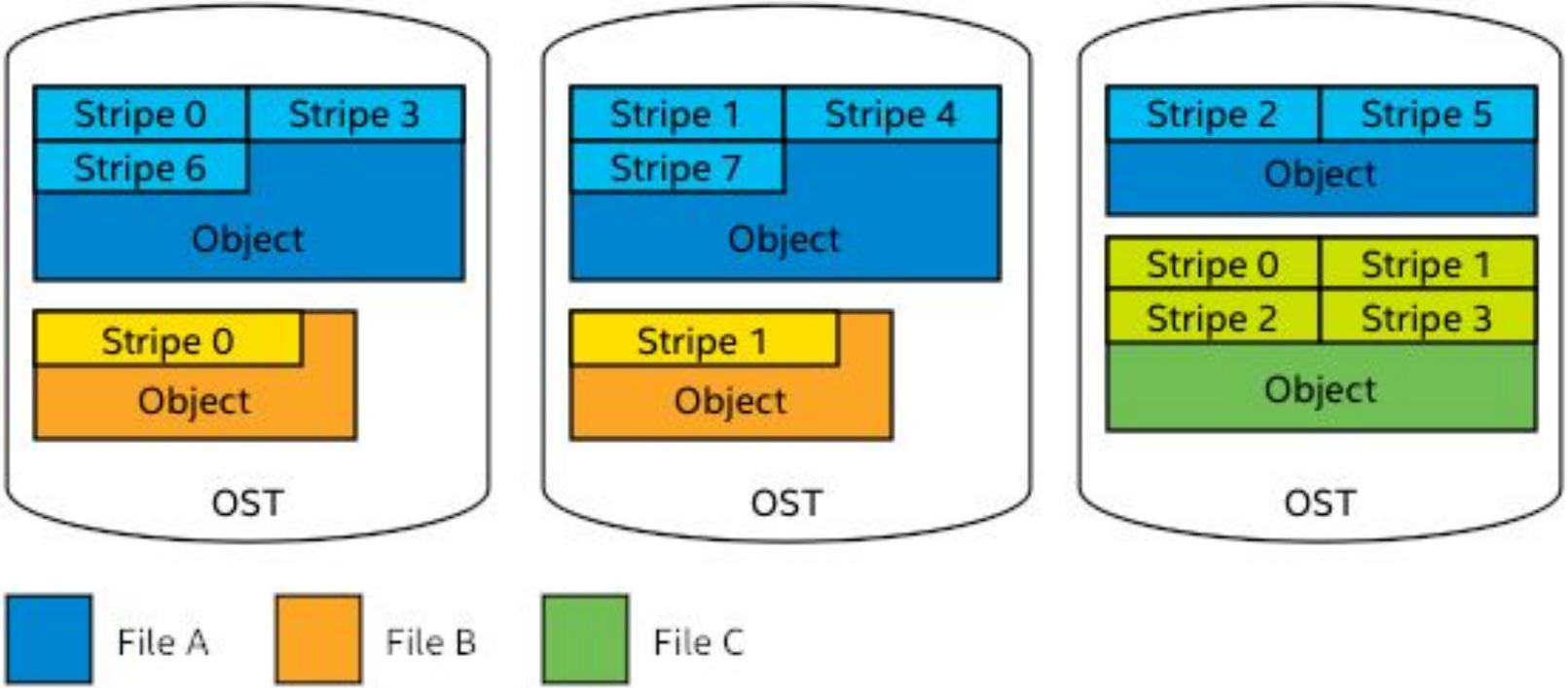
Reading data from one 16MB file on **lustre** is enormously faster than from 400 40KB files

- 👍 Sequential I/O of large datasets
- 👎 I/O of many small datasets or random I/O of large datasets

Lustre file system



File striping in Lustre FS



File striping in Lustre FS (cont.)

Lfs getstripe <file_name>

Lfs setstripe --stripe_size=1m --stripe_count=2 <file_name>

Use local disk for data I/O

Local disks on compute nodes are SSD

- /project or /scratch space → $SLURM_TMPDIR
- Loading data from $SLURM_TMPDIR
- Computing
- Saving intermediate/final results to $SLURM_TMPDIR
- $SLURM_TMPDIR → /project or /scratch space

Job Scheduler --- Slurm

- **sbatch abc.sh**: to submit a job script
- **squeue**: to list the status of submitted jobs
- **sacct** : to show details of recent jobs
- **scancel**: to kill jobs

Job script example (abc.sh)

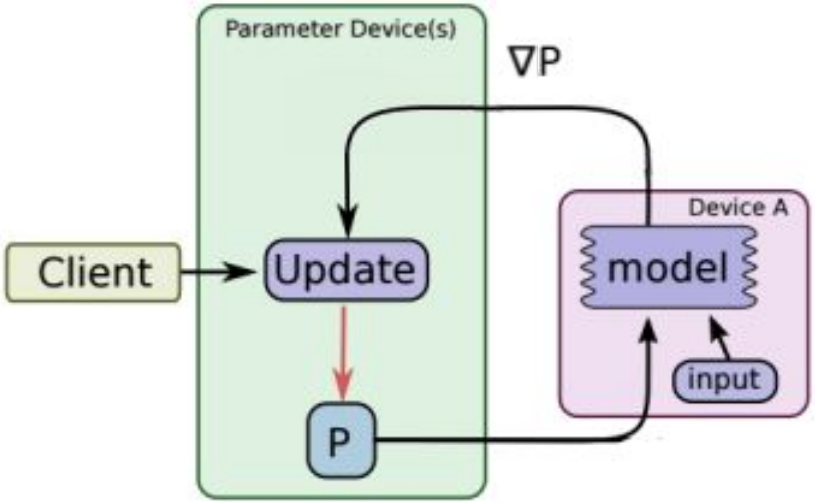
```
#!/bin/bash
#SBATCH --time=0-10:05          # Run time limit (DD-HH:MM)
#SBATCH --account=def-user      # account
#SBATCH --ntasks=4              # Number of MPI processes, default 1
#SBATCH --cpus-per-task=32      # Normally defined for threaded jobs
#SBATCH --gres=gpu:2            # number of GPUs
#SBATCH --mem=120G              # memory request

srum ./myprog
```

Distributed Parallel DNN training

- Overview
- Distributed DNN training on a single node
- Distributed DNN training across multiple nodes

NN training

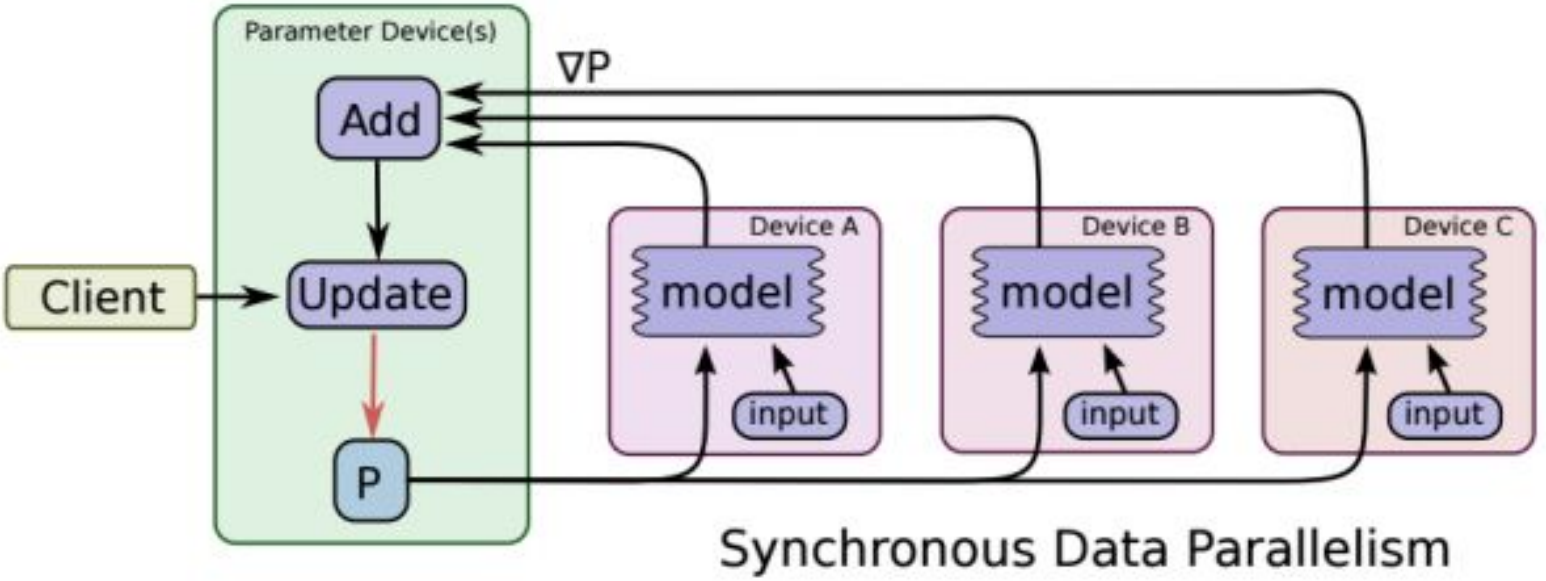
- Feed forward
- Back propagation
- Apply gradients to update parameters (or variables) of the model



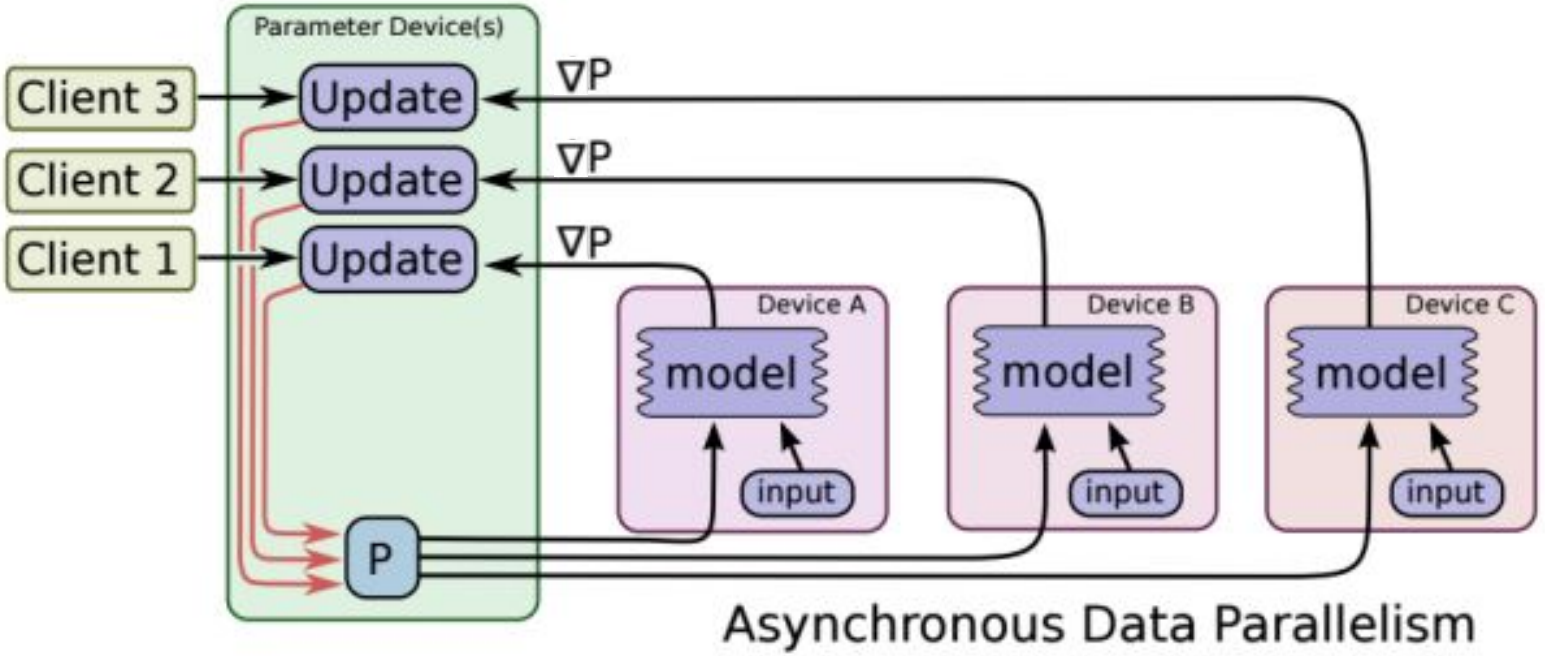
Two different parallelisms

- Data parallelism
  - Data are divided and each chunk is sent to each device
- Model parallelism
  - Model (graph) are divided, each partition is sent to each device

Synchronous data parallelism



Asynchronous data parallelism

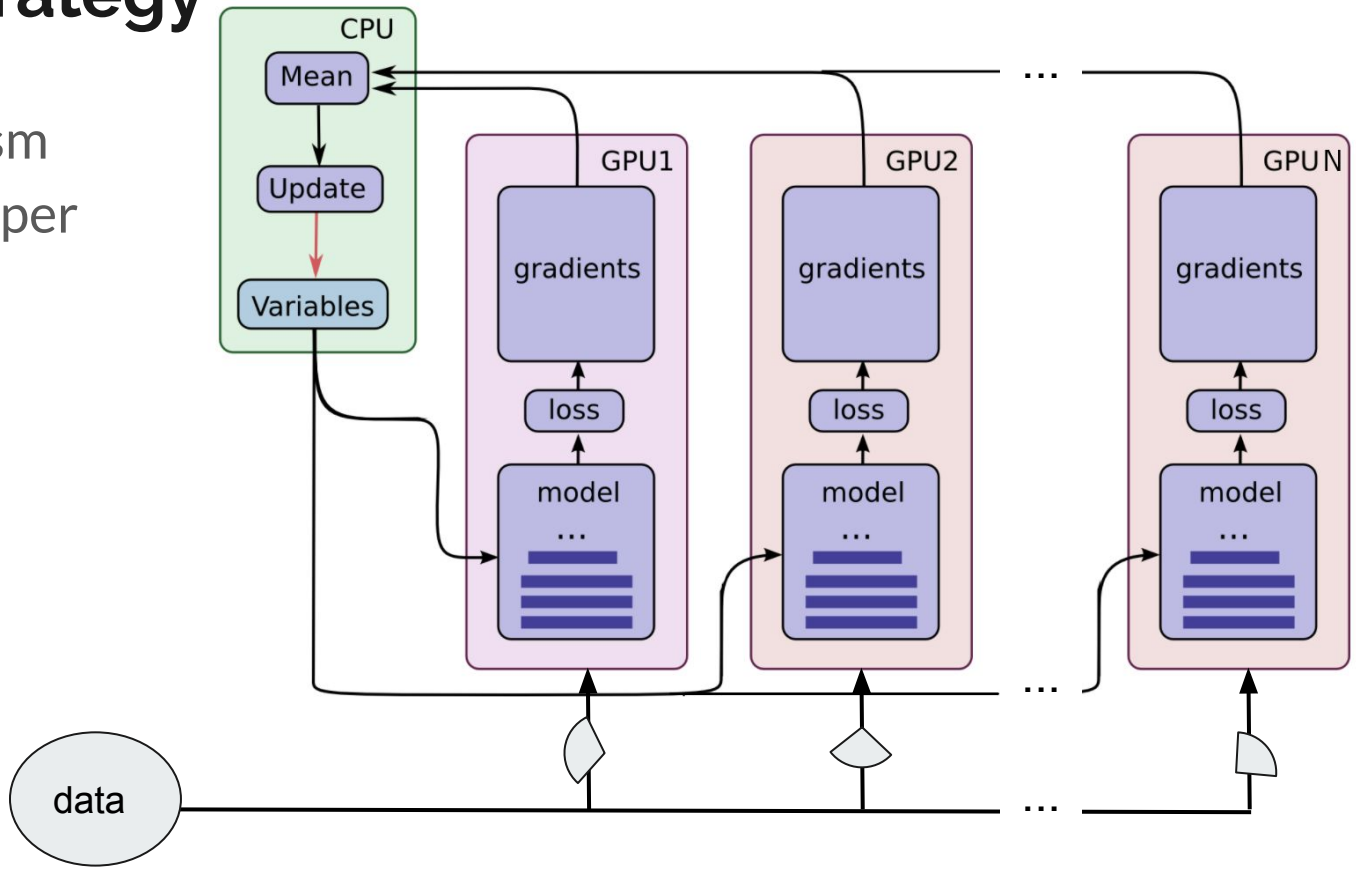


Overview of distributed strategies

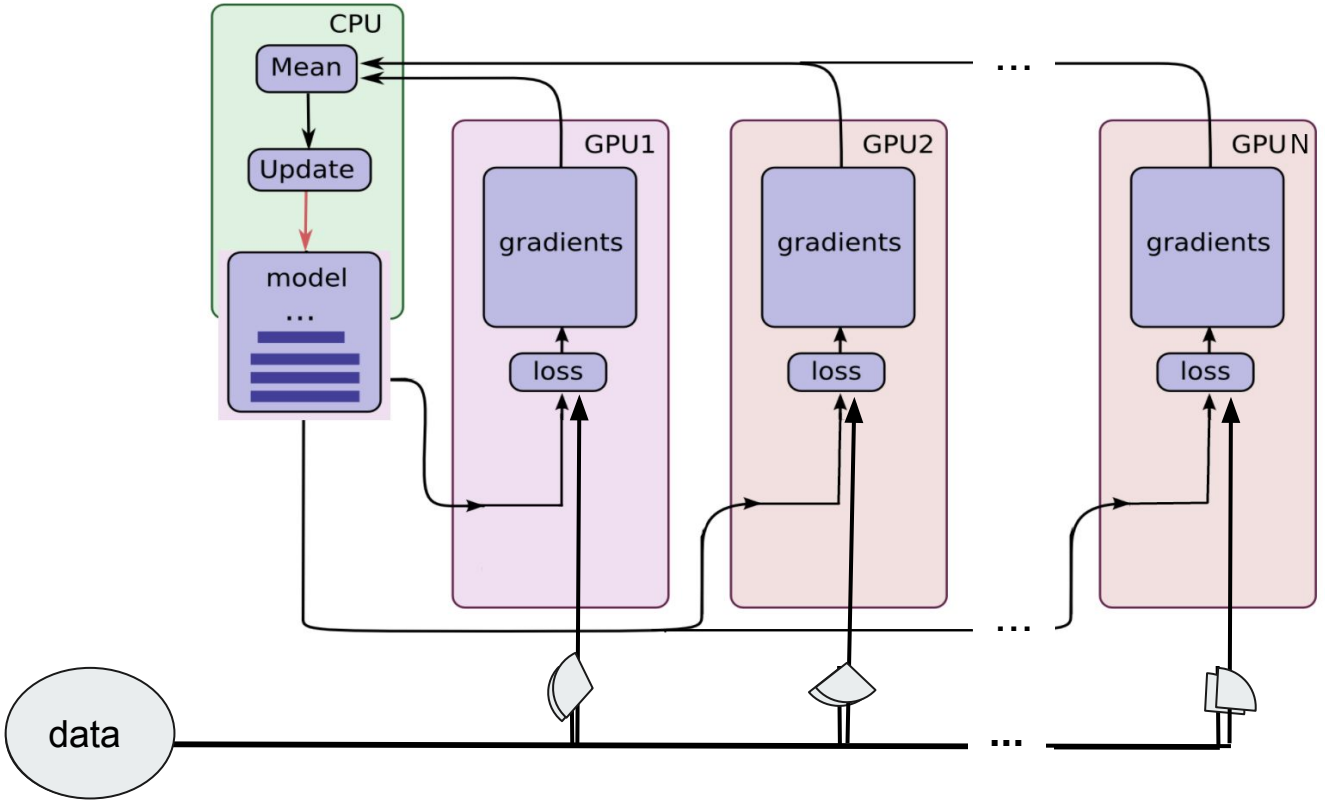
- Synchronous distributed training
  - Mirrored
  - Central storage
  - Multi-worker mirrored
  - TPU (same as Mirrored except for its own all-reduce)
- Asynchronous distributed training
  - Parameter server

Mirrored strategy

- Data parallelism
- Model replica per GPU



Central storage strategy



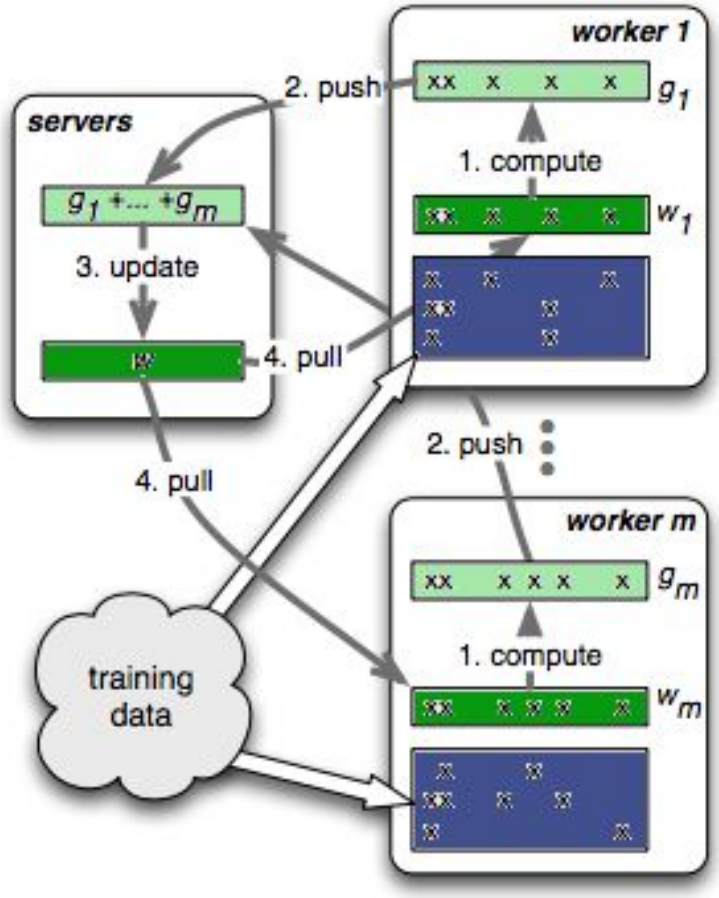
Parameter server strategy

Each worker

- Computes gradient on subset of image data
- Pushes the gradient to the server
- Pulls new parameter from server

Each server

- Aggregates gradients
- Updates parameters



Your choice of distributed NN training

- Using one of the above strategies
- Using custom distributed one:

```
with tf.device(“GPU:0”) :
        ……
        Code block that is run on GPU:0
        ……
……

with tf.device(“GPU:1”) :
        ……
        Code block that is run on GPU:1
        ……
```

The mirrored strategy

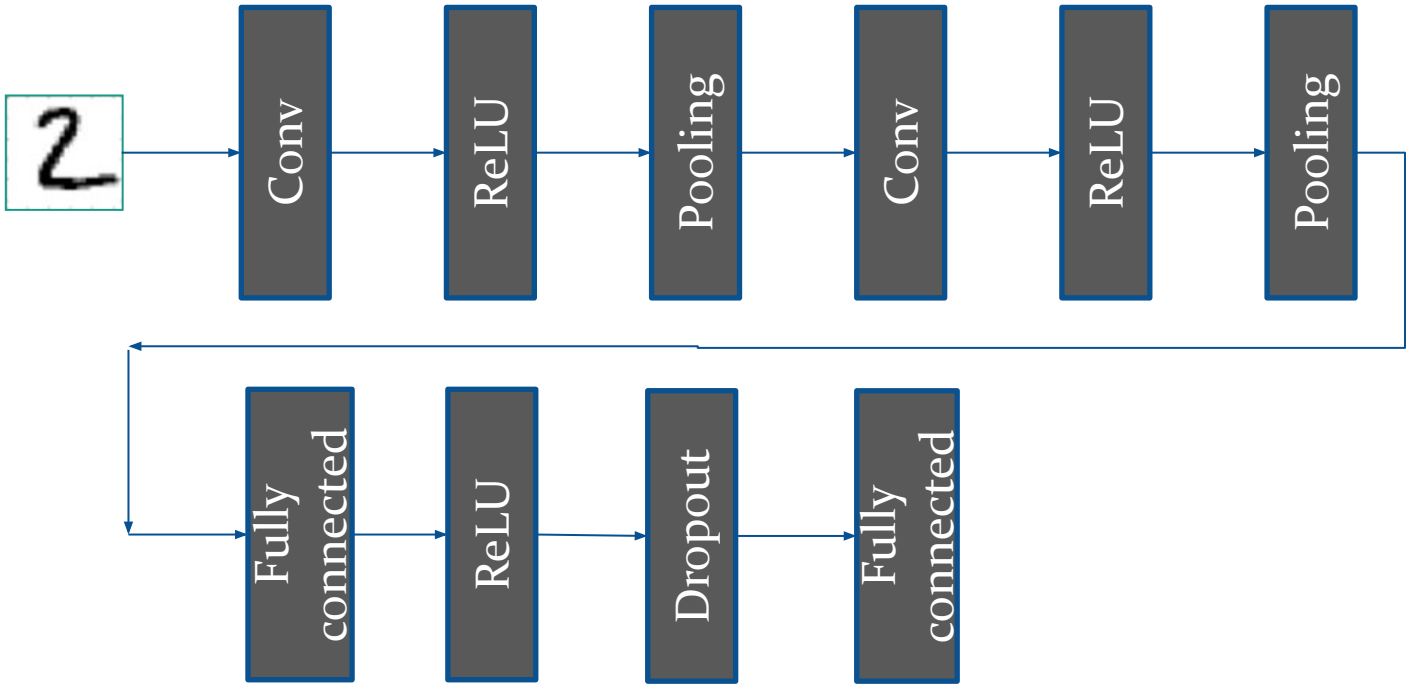
- How to create NN with TF's different APIs
- How to change them to use multiple GPUs on the same node
- Benchmarking
  - Multiple CPUs
  - Single GPU (p100, v100)
  - Multiple GPUs on single node
  - ~~Distributed multiple GPUs across multiple nodes~~

Benchmarking

- Tool: Tensorflow
  - 1.13
  - 2.0 Beta (tf_upgrade_v2 --infile tensorfoo.py --outfile tensorfoo-upgraded.py)
- Machine learning example
  - Task: Recognition of handwritten digits
  - Data: MNIST

Benchmarks on machine learning

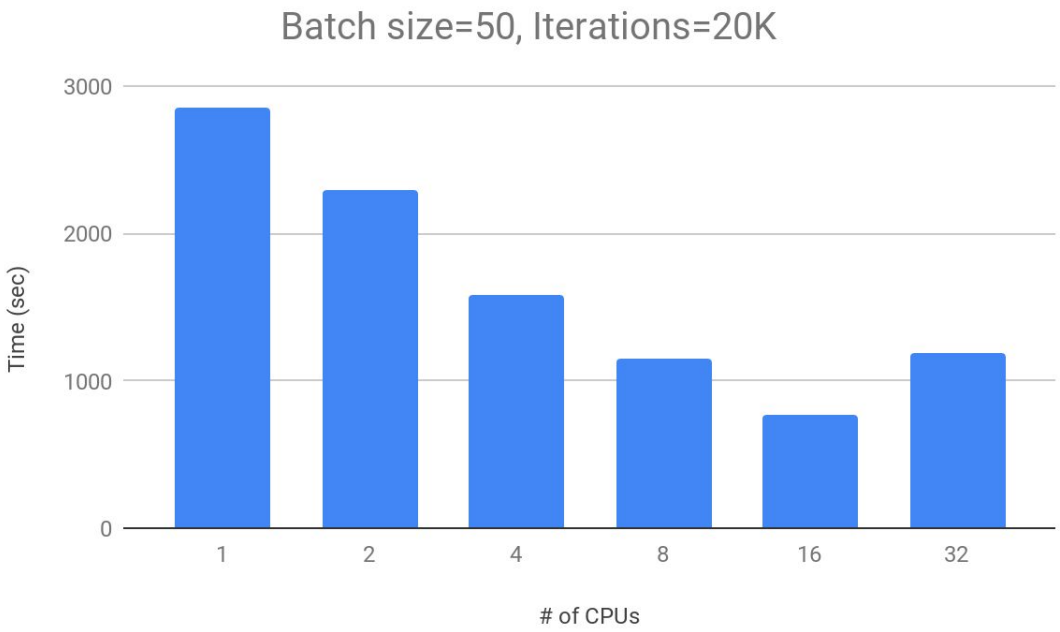
- Using recognition of mnist handwritten digits as an example



Handwritten digits recognition codes

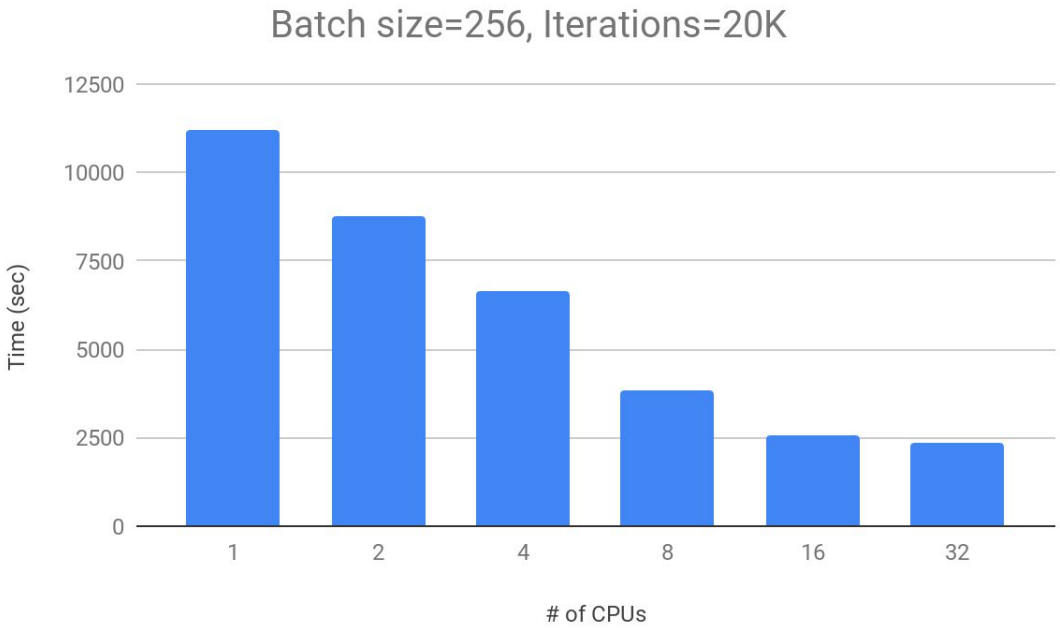
- low-level API
- Keras
- estimator

Benchmark of multiple CPUs



Batch size = 50 Iterations = 20000	
# of CPUs	Time (sec)
1	2854.43
2	2295.09
4	1582.72
8	1153.90
16	762.10
32	1187.92

Benchmark of multiple CPUs (cont.)



Batch size = 256 Iterations = 20000	
# of CPUs	Time (sec)
1	11193.94
2	8757.93
4	6645.16
8	3838.28
16	2559.46
32	2336.07

Benchmark of single GPU

Batch size = 256, Iterations = 20000

nVidia Pascal p100 GPU

- ~157 seconds (low-level API)
- ~133 seconds (Keras)
- ~185 seconds (estimator)

1 p100 GPU is ~15 times faster than 16 CPU cores (Intel E5-2683 v4 Broadwell)

Benchmark of single GPU (cont.)

Batch size = 256, Iterations = 20000

	nVidia Pascal p100 GPU	nVidia Volta v100 GPU
low-level API	157	103
Keras	135	128
Estimator	185	133

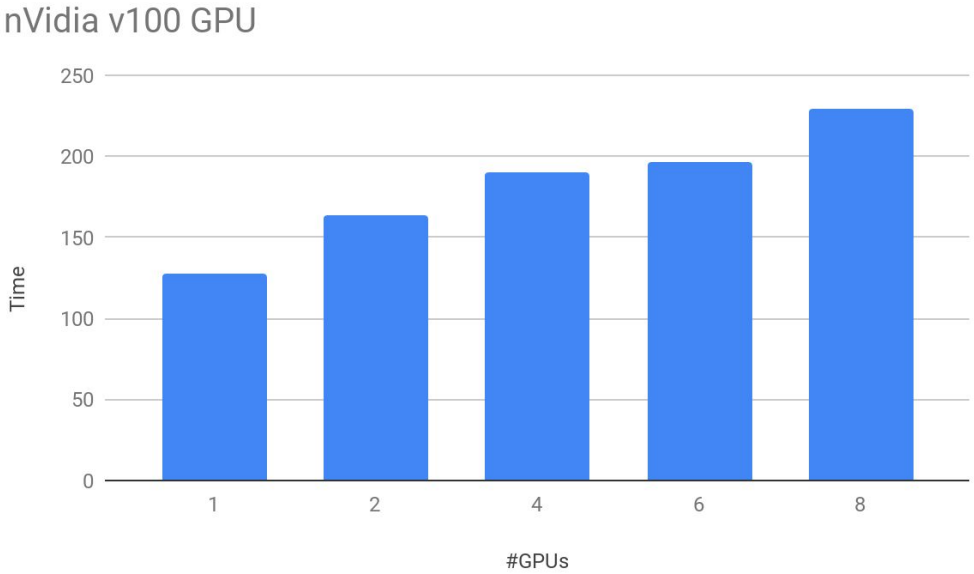
More GPUs we use,

the faster training will be?

Convert to Multi-GPU version

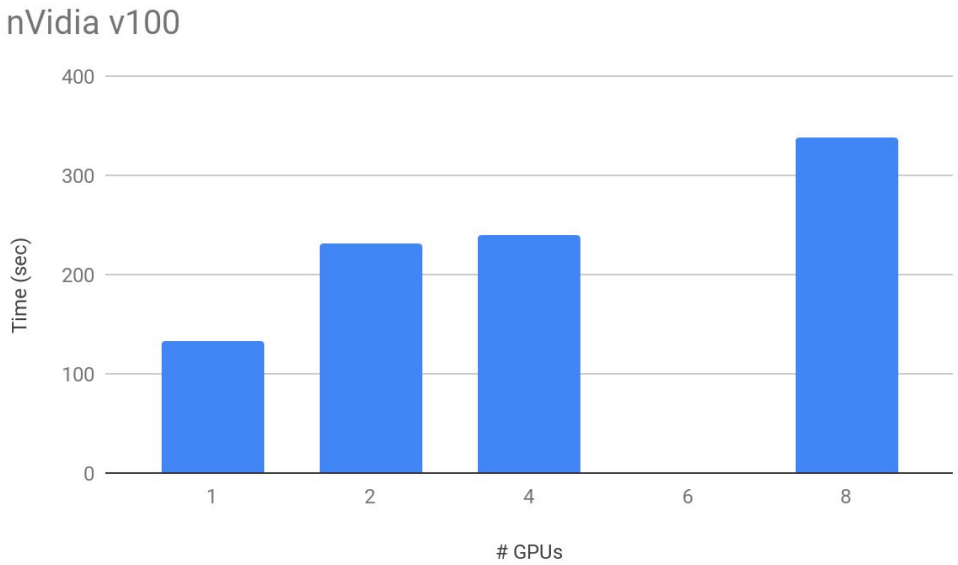
- low-level API
- Keras
- estimator

Benchmark of multiple GPUs (Keras)



Batch size = 256 Iterations = 20000		
# of GPUs	Time (sec)	
	p100	v100
1	135	128
2	161	164
4		190
6		196
8		229

Benchmark of multiple GPUs (Estimator)



Batch size = 256 Iterations = 20000		
# of GPUs	Time (sec)	
	p100	v100
1	185	133
2	252	231
4		239
6	*	*
8		338

* Batch_size must be multiple of #GPUs

More GPUs we use,

the faster training will be?

No! Not always. Why?

Why?

Hypothesis: The computing task of the MNIST example (~3 million trainable parameters) is far below the capacity of 1 GPU.

Test:

- Increase the scale of the NN
- Increase the batch size

Benchmark of multiple GPUs (Keras)

Batch size = 2048 Iterations = 20000	
# of GPUs	Time (sec)
	v100
1	343
2	301
4	269
8	268

Batch size = 8192 Iterations = 20000	
# of GPUs	Time (sec)
	v100
1	1203
2	873
4	856
8	828

Benchmark of multiple GPUs (Estimator)

Batch size = 2048 Iterations = 20000	
# of GPUs	Time (sec)
	v100
1	723
2	447
4	357
8	431

Batch size = 8192 Iterations = 20000	
# of GPUs	Time (sec)
	v100
1	2791
2	1246
4	818
8	714

Why?

Hypothesis: The computing task is far below the capacity of 1 GPU.

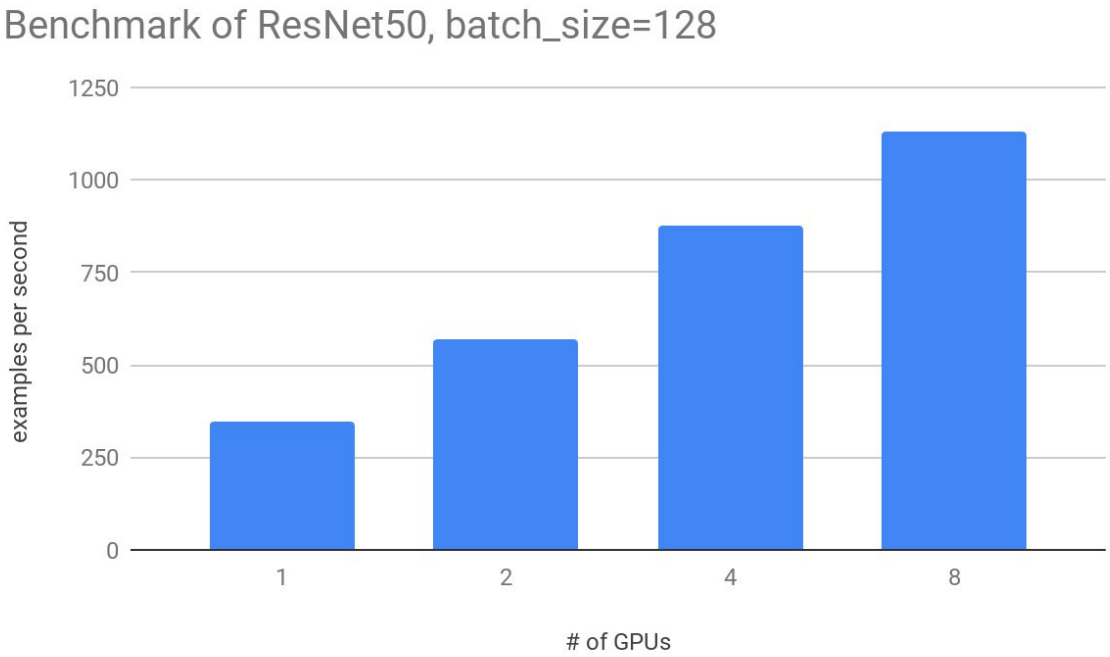
Test:

- Increase the scale of the NN
- Increase the batch size

ResNet-50

- 50 layers with default image size = 224 (23 million trainable parameters)
- git clone https://github.com/tensorflow/models.git
- Training parameters
  - Use synthetic data instead of real image data
  - Batch_size = 128
  - Number of GPUs = 1, 2, 4, 8

Benchmark of ResNet-50



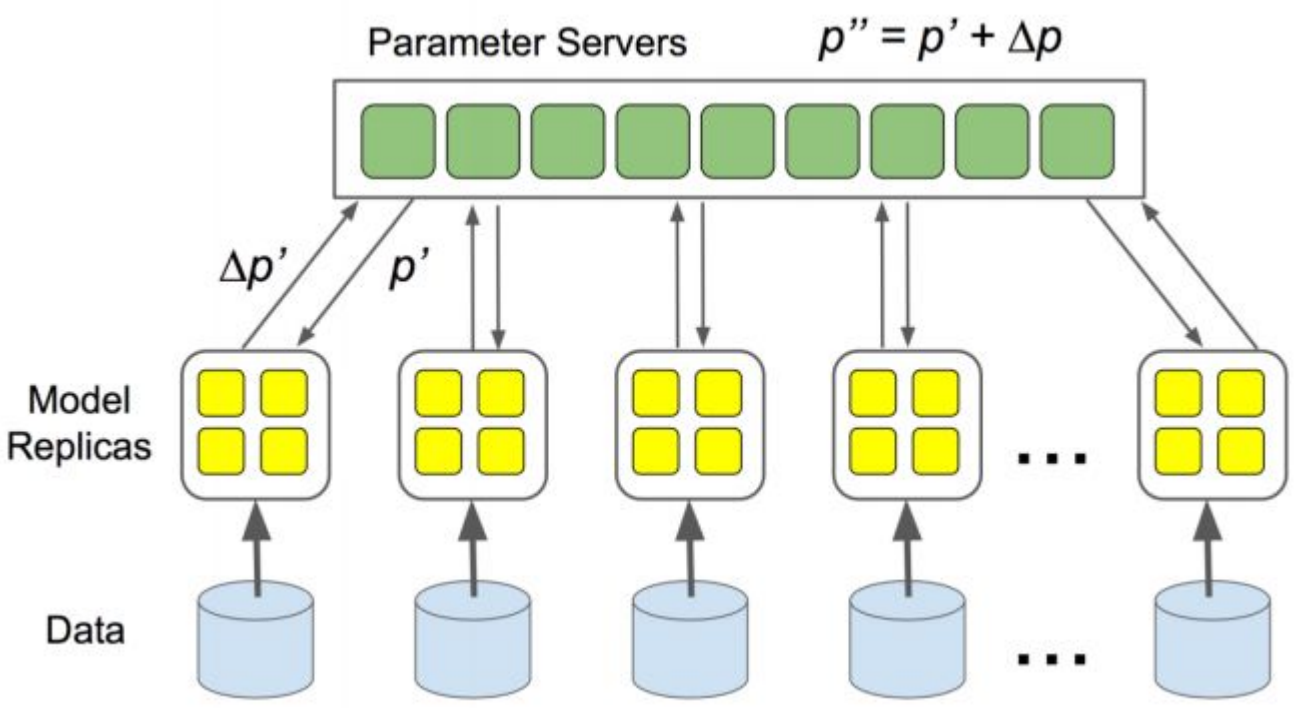
Batch size = 128	
# of GPUs	Examples/sec
	v100
1	345
2	567
4	877
8	1132

Orchestra in play

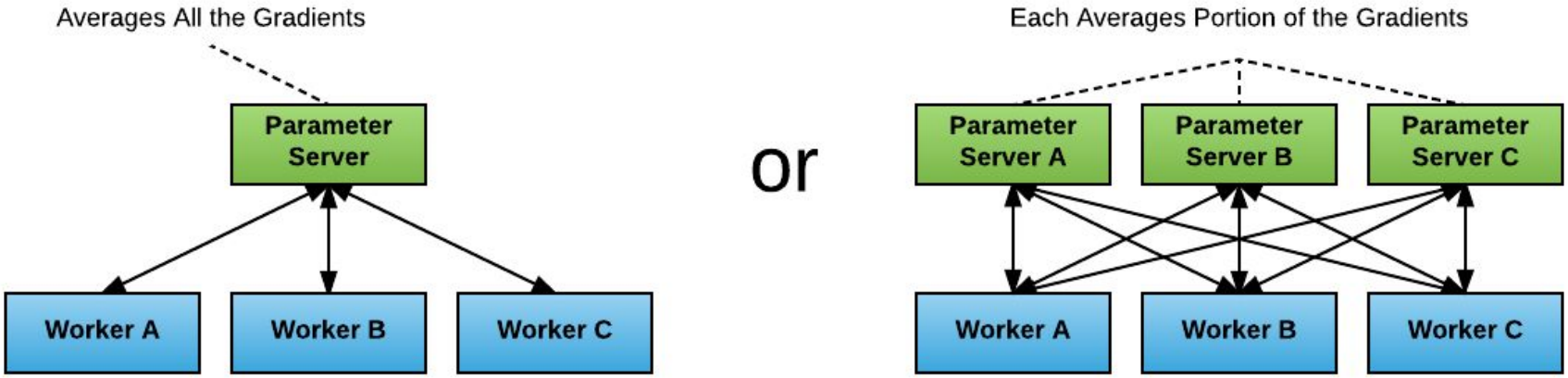
Horovod --- Multi-GPU multi-node parallelism by Uber AI team

- https://github.com/uber/horovod
- TF's MultiWorkerMirroredStrategy

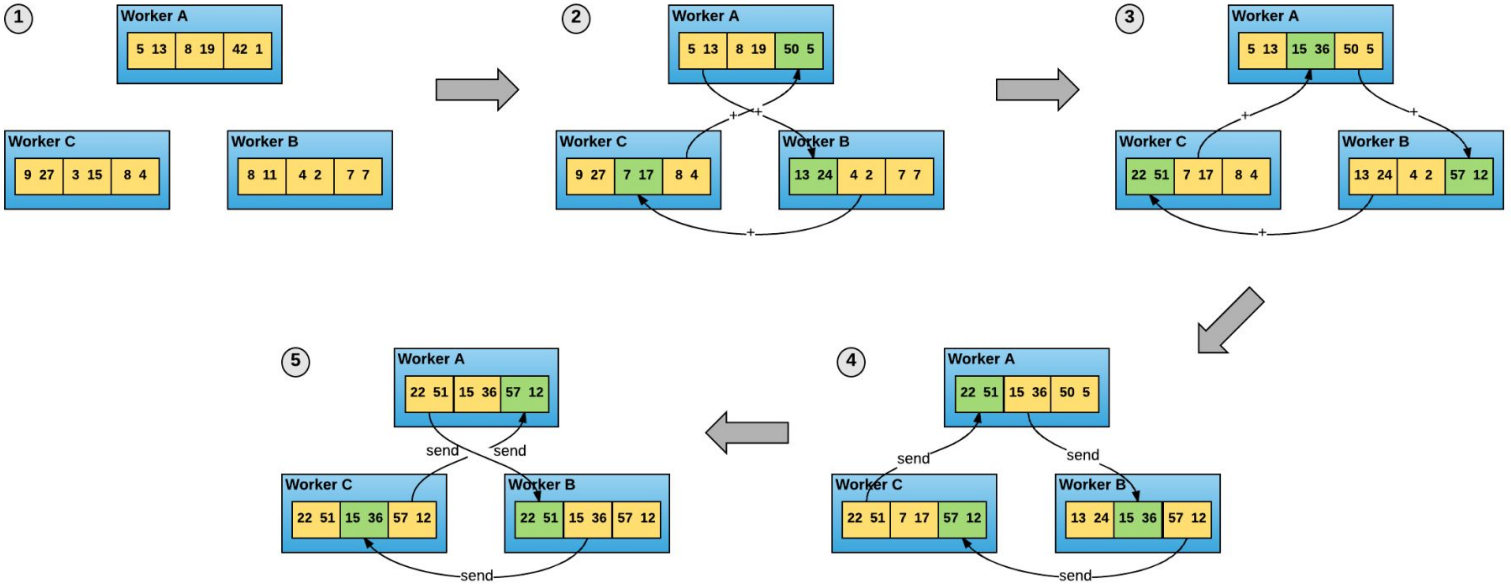
Horovod --- Multi-GPU Multi-node parallelism



Ratio of workers vs parameter servers

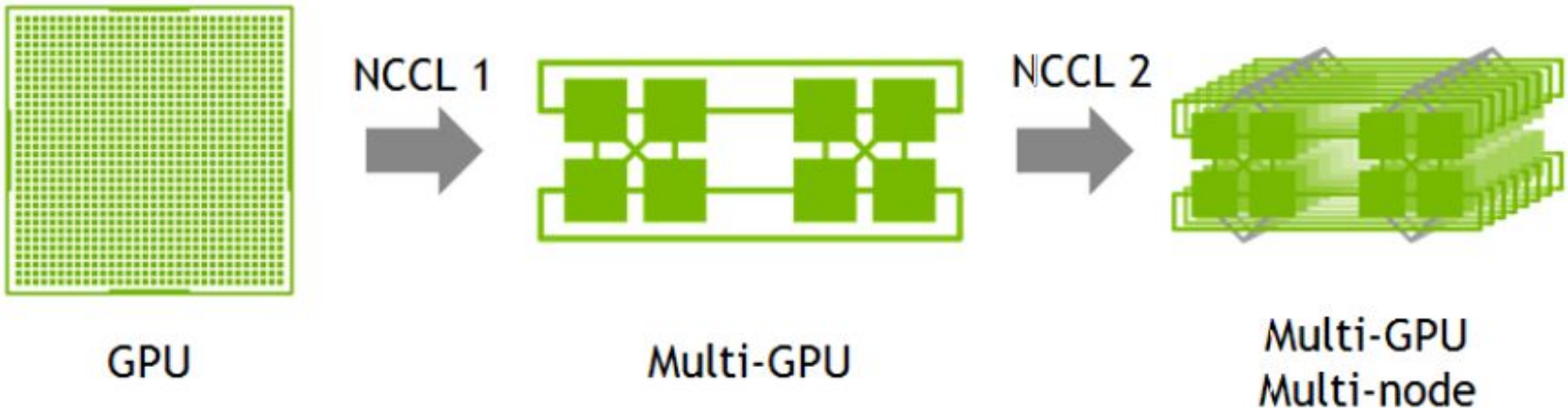


Baidu's ring-allreduce (2017)



NCCL ring-allreduce

- NCCL 1: ring-allreduce across multiple GPUs on a node
- NCCL 2: ring-allreduce across multiple nodes



Tensor fusion

Combination of multiple small tensors into a big one

Horovod --- Multiple GPUs on multiple nodes

- Load python modules
  - module load python/3.7
  - Module load cuda cudnn
  - module load scipy-stack/2019a
- Create python virtual environment
  - export ENV=$HOME/nextai-env
  - virtualenv --no-download $ENV
  - source $ENV/bin/activate
  - pip install --upgrade pip

Horovod --- Multiple GPUs on multiple nodes (cont.)

- Install Tensorflow in the virtual environment
  - pip install tensorflow_gpu --no-index
- Download/install NCCL2
  - https://developer.nvidia.com/nccl/nccl-download
  - setrpaths.sh --path /path/to/nccl2 --add_origin
- Install Horovod in the virtual environment
  - export HOROVOD_CUDA_HOME=$CUDA_HOME
  - export HOROVOD_NCCL_HOME=/path/to/nccl2
  - export HOROVOD_GPU_ALLREDUCE=NCCL
  - pip install --no-cache-dir horovod

Horovod --- Multiple GPUs on multiple nodes (cont.)

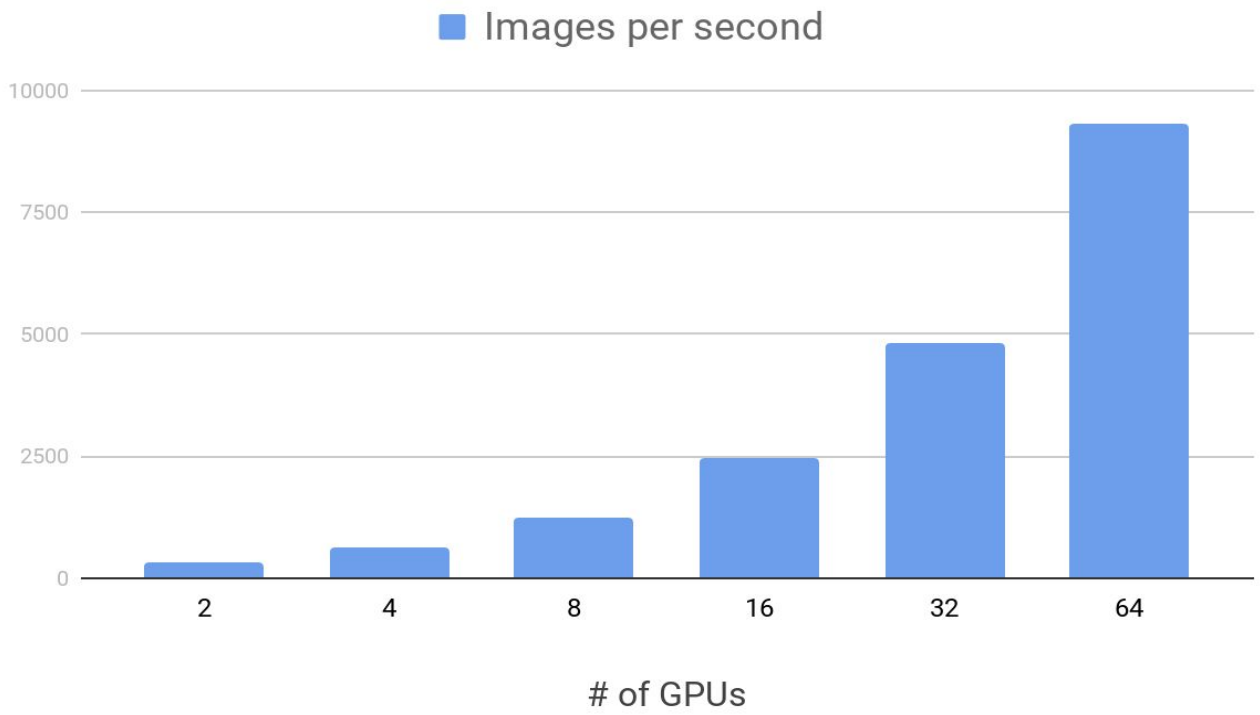
- Clone TF benchmark repo:
  - git clone https://github.com/tensorflow/benchmarks.git
- Run the benchmarking script
  - mpirun -np $((SLURM_NTASKS/16)) -bind-to core -map-by slot:PE=16 -report-bindings -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl ^openib python -u scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model resnet50 --batch_size 32 --variable_update horovod

Benchmarks with ResNet50

Images/sec: (ResNet-50 with batch size=32 per GPU)

# of Nodes	# of GPUs	Images per second
1	2	322.93
2	4	629.01
4	8	1238.57
8	16	2443.49
16	32	4825.47
32	64	9318.20 (10143.52)

Benchmarks with ResNet50



Thank you!

Any questions?