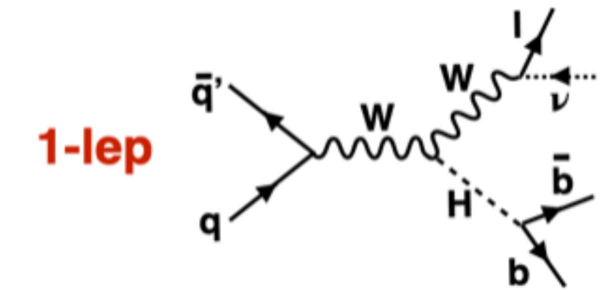
User experiences: Deep Sets neural network with different job scheduling

Fang-Ying Tsai

June 10 2022, NPPS meeting

Supervisor & advisors
John Hobbs, Giacinto Piacquadio,
Torre Wenaus, Alexei Klimentov

Recap



- Estimate modeling uncertainties for the VHbb analysis.
  - 3 channels in the VHbb studies: 0 lepton ($Z \rightarrow \nu\nu$), **1 lepton** ($W \rightarrow l\nu$), 2 leptons($Z \rightarrow ll$) where l = e, μ
  - Main backgrounds: ttbar, **V+jets**, multijets, single top, Diboson.
- V+jets MC modeling shape systematic is described through nuisance parameters in the fit for the signal strength, $\mu_{VH}^{b\bar{b}}$, measurement.
  - kinematic reweighting based on 1-dim variable: pTV and mBB are not enough to cover all shape systematics in some kinematics.
  - Need more sophisticated algorithms: BDT and DSNN are compared in this study.
- In the DSNN, we aim to have a generic classification for VH(bb) as well as VH(cc) analyses in both boosted and resolved regimes.
  - Instead of using higher level input variables (e.g. mBB), DSNN uses the 4-vectors of the final state particles as training inputs.
  - the BDT used in the VH(bb) analysis is very analysis specific.
- Computing Challenges.

DSNN computing needs

- **Pre-processing** (Ntuple -> Numpy)

	Sherpa (events)	MGPy8 (events)	Time for getting numpy arrays	MaxRSS
MCa	25053101	7570460	02:32:43	27 GB
MCd	30415633	9017054	02:54:44	139 GB
MCe	40936533	11854840	05:06:13	186 GB
Total	96,405,267	28,442,354	x	>250 GB

- **Training** TensorFlow model using GPU: about 94GB memory per node.

Ray Tune's progress reporter table

```
== Status ==
Current time: 2021-12-07 21:44:28 (running for 00:46:30.21)
Memory usage on this node: 94.5/375.9 GiB
Using FIFO scheduling algorithm.
Resources requested: 4.0/320 CPUs, 4.0/4 GPUs, 0.0/968.92 GiB heap, 0.0/419.24 GiB objects (0.0/4.0 accelerator_type:V100)
Result logdir: /global/homes/f/ftsai/ray_results/train_and_score_2021-12-07_20-57-57
Number of trials: 4/4 (4 RUNNING)
+---------------------------+----------+---------------------+-------------------+-------------------+
| Trial name                | status   | loc                 | F_sizes           | Phi_sizes         |
|---------------------------+----------+---------------------+-------------------+-------------------|
| train_and_score_693c6_00000 | RUNNING  | 128.55.144.58:68695 | (120, 120, 120) | (120, 120, 100) |
| train_and_score_693c6_00001 | RUNNING  | 128.55.144.61:28503 | (125, 125, 125) | (120, 120, 100) |
| train_and_score_693c6_00002 | RUNNING  | 128.55.144.60:28067 | (120, 120, 120) | (125, 125, 100) |
| train_and_score_693c6_00003 | RUNNING  | 128.55.144.59:19337 | (125, 125, 125) | (125, 125, 100) |
+---------------------------+----------+---------------------+-------------------+-------------------+
```

(required 4 nodes for the DSNN training using Ray clusters.)

Computing resources options

- **Part 1.** The DSNN has been running on…

Job Scheduler	Environment	Where	CPU	GPU
HTCondor High Throughput Computing	1. venv module 2. Docker (python virtualenv, docker)	lxplus	1 CPU/2GB RAM 32 CPU/node	V100, T4
slurm workload manager	Conda (python CONDA)	BNL	250GB/node RAM	K80, P100
slurm workload manager	Shifter+Docker (SHIFTER, docker)	NERSC	Haswell (125GB/node) KNL (96GB/node)	V100
kubernetes	Docker (docker)	Google cloud	scalable	T4, P100, K80
PanDA	Docker (docker)	GRID	32GB/site	Available, but I didn't test it yet.

- **Part 2.** Speeding up. The DSNN has been tested with DASK (on GCP) and Ray (on NERSC) clusters to make data preprocessing and training go faster.

DASK RAY

HTCondor

Job Scheduler	Environment	Where
HTCondor	1. venv module 2. Docker	lxplus

Easy to share the environment

HTCondor Job description file (JDF)

```
universe                = docker
docker_image           = fyingtsai/dsnnr_4gpu:latest
executable              = myDSNNr_run.sh
arguments               = /etc/hosts
transfer_input_files    = data.tar.gz, train.py,DSNNr_lib.py, DSNNrENV.tar.gz
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT
output                  = log/$(ClusterId).$(ProcId).out
error                   = log/$(ClusterId).$(ProcId).err
log                     = log/$(ClusterId).$(ProcId).log
request_memory          = 5G
request_gpus            =  1
+Requirements           = OpSysAndVer =?= "CentOS7"
queue
```

The Docker universe feature

The venv module

setup virtual python environment:
python3 -m **venv** DSNNrENV
source DSNNrENV/bin/activate
[DSNNrENV] pip install -r requirement.txt

absl-py==0.11.0
astunparse==1.6.3
awkward==1.1.1
cachetools==4.2.1
certifi==2020.12.5
chardet==4.0.0
……

- I can ask for more memory (~10GB or so), but this job is going to wait a long long time to be executed.
- Members of the accounting group group_u_BE.ABP.NORMAL should have access to run on nodes with large memory (1T).

Docker

docker

Job Scheduler	Environment	Where
HTCondor	1. venv module 2. Docker	lxplus

Easy to share the environment

- Best practices for writing Dockerfiles if you want to build an efficient images properly, here.

A sample Dockerfile

```
# Our base image
FROM tensorflow/tensorflow:latest-gpu

# Some common environmental variables that Python uses
ENV LANG=C.UTF-8 LC_ALL=C.UTF-8

# Install lower level dependencies
RUN apt-get update --fix-missing && \
    apt-get install -y curl python3 python3-pip &&  \
    update-alternatives --install /usr/bin/python python /usr/bin/python3 10 && \
    update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 10 && \
    apt-get clean && \
    apt-get autoremove && \
    rm -rf /var/lib/apt/lists/*

# Install a specific version of TensorFlow
# You may also install anything else from pip like this
#RUN pip install --no-cache-dir tensorflow-gpu==1.12.0
RUN pip install --upgrade pip
RUN pip install numpy
RUN pip install tensorflow
RUN pip install energyflow
RUN pip install uproot
RUN pip install matplotlib
RUN pip install sklearn
RUN pip install awkward
RUN pip install pandas
```

- Docker build (doc)

```
$ docker build -t <tag name> -
< Dockerfile
```

```
KellyTsai@Kellys-MacBook-Pro:Docker$ docker build -t fyingtsai/tensorflow-4gpu - < DockerFile
Sending build context to Docker daemon  2.56 kB
Step 1/3 : FROM tensorflow/tensorflow:latest-gpu
 ---> 13347da1cd62
Step 2/3 : ENV LANG C.UTF-8 LC_ALL C.UTF-8
 ---> Using cache
 ---> b354ad767f6d
Step 3/3 : RUN apt-get update --fix-missing &&     apt-get install -y curl python3 python3-pip &&     update-alternatives --install /usr/bin/python python /usr/bin/python3 10 &&     update-alternatives --install /usr/bin/pip pip /usr/bin/pip3 10 &&     apt-get clean &&     apt-get autoremove &&     rm -rf /var/lib/apt/lists/*
 ---> Using cache
 ---> 21fc28b9b7d5
Successfully built 21fc28b9b7d5
```

- push the image to my docker hub through *docker commit*, *docker login*, and *docker push* etc. commands.
  - fyingtsai/dsnnr_4gpu:latest (for TensorFlow gpu)

Slurm (1)

slurm
workload manager

Job Scheduler	Environment	Where
Slurm	Conda (like pipenv)	BNL

Docker currently has multiple design points that make it unfriendly to HPC systems. The issue that usually stops most sites from using Docker is the requirement of "only trusted users should be allowed to control your Docker daemon" [Docker Security] which is not acceptable to most HPC systems.

Slurm sbatch description

- Definitely not easy for a beginner to submit his/her first Slurm job using GPUs. (I had major help from Doug Benjamin and SDCC supports!)

```
#!/bin/bash
#SBATCH --partition usatlas
#SBATCH --time=24:00:00
#SBATCH --account=tier3
#SBATCH --nodes=1
#SBATCH --qos usatlas
#SBATCH --gres=gpu:1
#SBATCH --mem=230000
eval "$(/hpcgpfs01/software/anaconda3/2020-11/bin/conda shell.bash hook)"
conda create --prefix ./tf2-gpu tensorflow-gpu matplotlib tensorboard
conda activate /hpcgpfs01/scratch/ftsai/DSNNrBranch/tf2-gpu
conda install -c conda-forge keras=2.4.3 ....
pip install energyflow root-numpy
pip install...
srun ./myDSNNr_run.sh
```

high memory usage, but jobs were executed all immediately.

1. Setup the Anaconda environment: using this eval script to avoid messing with the .bashrc file

2. Create the new condo python environment

3. Activate the environment

Note to make the shell prompt shorter I followed these instructions – [https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html - specifying-location](https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html)
conda config --set env_prompt '({name})'
This works after the environment is reactivated the next time.

- Finished my DSNN studies (data preprocessing & training) all in this way, so nothing can complain after going through the setup.

Slurm (2)

Job Scheduler	Environment	Where
Slurm	Shifter + Docker	NERSC

- Shifter developed by NERSC brings containers to HPC.
  - Intro to Shifter, here. How to use, here.
- Much easier. Have tested it successfully with the DSNN training.

Slurm sbatch description

```
#!/bin/bash
#SBATCH --account=m2616
#SBATCH --time=10:00:00
#SBATCH --nodes=2
#SBATCH -C gpu
#SBATCH -o joblogs/%j.out
#SBATCH -e joblogs/%j.err
#SBATCH --image=fyingtsai/dsnnr_4gpu:latest
#SBATCH -G 2
#module load cuda/10.1.243
nvidia-smi
srun shifter python examples/myDSNNr_train.py
```

docker image option (better to be experienced with Docker so you can make a proper image with env. e.g. deploying conda env in Docker.)

Shifter command

Google Kubernetes Engine (GKE)

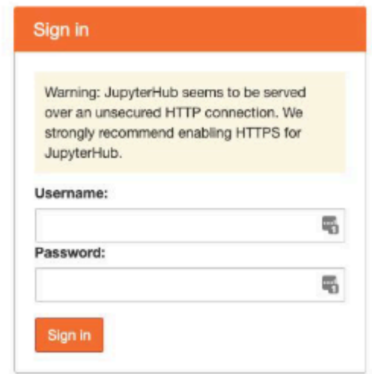
Fernando’s slides, here

- No setup pain.
  - Jupyter and Dask has been launched on kubernetes cluster.
  - The docker images have been deployed in GCP.

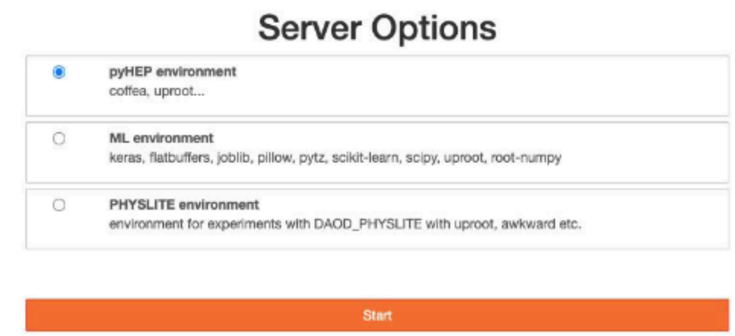
- Enjoy the power of parallelism that DASK provides.
  - Just requires some experiences to know where and with what DASK APIs to deploy the DASK cluster in my ML framework.
  - Best practices, here.

JupyterHub

http://jupyter.gcp4hep.org/



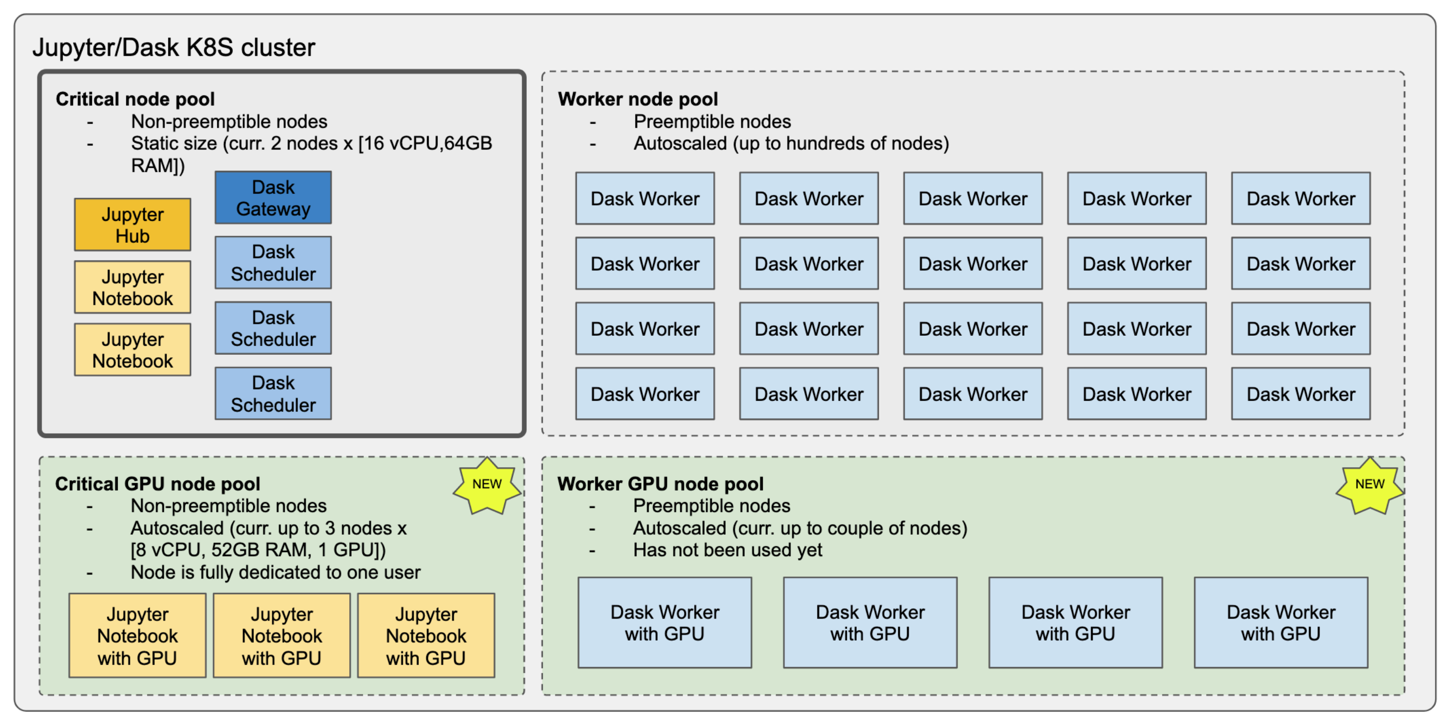
- Local accounts. I need to add new users
- Integration with other identity providers possible if long-term project



- Available images
  - pyHEP environment: dependencies suggested in this tutorial
  - ML environment: dependencies requested by Fang-Ying
  - PHYSLITE environment: image provided by Nikolai
- Images hosted in GCP Container Registry
- CVMFS available on notebooks

14

Jupyter/Dask cluster pools on Google



DSNN with DASK clusters

- Speeding up **data pre-processing** using DASK on GCP.
  - The **parallel** data processing can be done in 5-6 mins (the final amount of memory occupied ~ 47.4GB), while the standard data processing takes at least 40 mins.
  - Implemented DASK APIs such as client.scatter, dask.delayed, dask.compute etc.

How I use DASK APIs in the DSNN:

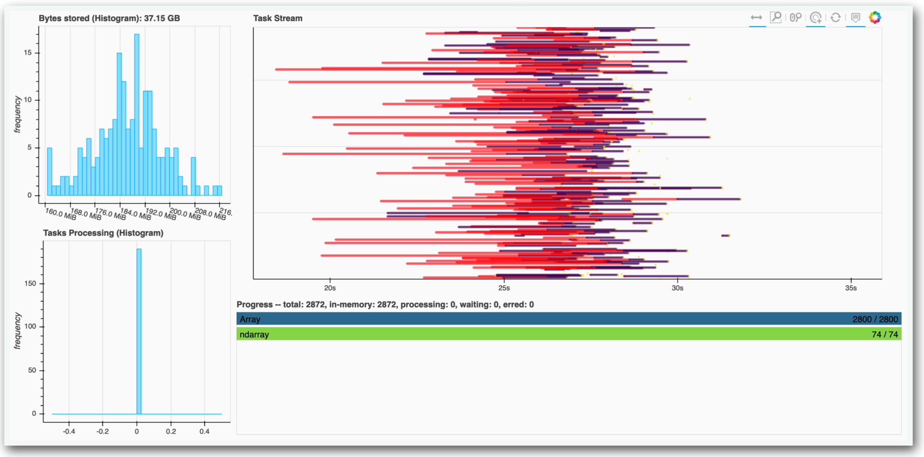
Code   Python 3 (ipykernel)

```
[18]: %%time
# def preprocess_data(MCa_tree,MCb_tree,){
features = ["TruthObj_px","TruthObj_py","TruthObj_pz","TruthObj_E","TruthObj_pdgid"]
# # MCa_tree = MCa_file["Tree_1Lep"]
# # MCb_tree = MCb_file["Tree_1Lep"]
delayed_MCaprocess = []
delayed_MCbprocess = []
delayed_MCaWeight = []
delayed_MCbWeight = []
for i, url in enumerate(url_array):
    nMCaEvents = myTree[i].num_entries
    nMCbEvents = myTree2[i].num_entries

    chunks = 10
    ChunkDivMCa = math.ceil(nMCaEvents/chunks)
    ChunkDivMCb = math.ceil(nMCbEvents/chunks)
    chunkMCa_tree=[]
    chunkMCb_tree=[]

    for idx,  chunk_arrayMCa in enumerate(uproot.iterate(myTree[i], step_size=ChunkDivMCa)):
        chunkMCa_tree.append(client.scatter(chunk_arrayMCa))
        dfs_delayed_MC= dask.delayed(preprocess_data_MC)(chunkMCa_tree[idx], features, 45)
        delayed_MCaprocess.append(dfs_delayed_MC[0])
        delayed_MCaWeight.append(dfs_delayed_MC[1])
    for idx,  chunk_arrayMCb in enumerate(uproot.iterate(myTree2[i], step_size=ChunkDivMCb)):
        chunkMCb_tree.append(client.scatter(chunk_arrayMCb))
        dfs_delayed_MC= dask.delayed(preprocess_data_MC)(chunkMCb_tree[idx], features, 45)
        delayed_MCbprocess.append(dfs_delayed_MC[0])
        delayed_MCbWeight.append(dfs_delayed_MC[1])

dfs_MCaprocess,  dfs_MCaWeight= dask.compute(delayed_MCaprocess, delayed_MCaWeight)
dfs_MCbprocess,  dfs_MCbWeight= dask.compute(delayed_MCbprocess, delayed_MCbWeight)
```

.scatter() & .delayed()

.compute ()

can monitor the progress in the DASK dashboard.



DSNN with Ray clusters RAY NERSC

- Speeding up **hyper-parameter tuning** using Ray APIs on NERSC computing facilities.
  - I asked for 4 nodes, 32 tasks, and 32 GPUs to test the parallel training with full Sherpa+MGPy8 datasets. To scan 4 combination of hyper-parameters, the job was done in 54.7 mins using Ray clusters, while w/o distributed computing, the same size of inputs was done in 87.8 mins.
  - Failed to use DASK to train TF with larger datasets in parallel on GCP.
- All I need to do is to follow the sample code, set up nodes (1 for the head and the rest for workers) and submit Slurm script. I used Slurm + Conda.
  - The only problem which made me get stuck was importing tensorflow took like forever.
    - -> This was solved by upgrading the TF version to 2.7 from 2.4.

Conclusions

- ML projects consist different needs in terms of resource requirements and libraries to run on them.
  - I'm satisfied with the slurm+conda system, but there is no scaling capability, and setting up conda is complex for a beginner (alternative: shifter.)
  - My htcondor requests on lxplus often wait in queues.
  - Kubernetes on GCP might be the easiest one for everyone.

- There are many tools out there for AI/ML that I haven't tried it out.
  - e.g. TensorFlow with Parquet files (may achieve CPU utilization efficiently in the process of data preprocessing.)
  - Heterogeneous system (e.g. Perlmutter@NERSC?)

- Other useful US ATLAS facilities@Chicago
  - https://indico.cern.ch/event/1135275/
  - I have a very good experience with htcondor here (running the track overlay framework though.)

Backups

Big PanDA

Job Scheduler	Environment	Where
Panda	Docker	GRID

- Submit jobs through PanDA queue.
  - The only challenge was I want to run two datasets within a job. I found the --secondaryDSs looks good for this purpose, and it does work nicely until I want to ask for more than 1 file.
- The problem was solved by using --notExpandSecDS and --notExpandInDS ((keep Number of input files >= nJob * nFilesPerJob, then the job won't be split) that was answered by Tadashi and PanDA dev people. Panda intro, here.
- The final arguments are:

```
 prun --containerImage docker://sjiggins/tensorflow-gpu-dsnnr:v1\
--exec="./myDSNNr_run.sh '%IN' '%IN2' '%IN3'"\
   --inDS user.sjiggins.mc15_13TeV.410470.PhPy8EG_A14_ttbar_hdamp258p75_nonallhad.evgen.EVNT.e6337.VHbb_DSNNr_ttbar-v3_1_Lep\
     --secondaryDSs IN2:3:user.sjiggins.mc15_13TeV.
 410464.aMcAtNloPy8EvtGen_ttbar_noShWe_SingleLep.evgen.EVNT.e6762.VHbb_DSNNr_ttbar-v3_1_Lep/,IN3:3:user.sjiggins.mc15_13TeV.
 410465.aMcAtNloPy8EvtGen_A14N23LO_ttbar_noShWe_dil.evgen.EVNT.e6762.VHbb_DSNNr_ttbar-v3_1_Lep/\
     --excludeFile data \
     --site=GOOGLE100 \
     --destSE=GOOGLE_EU\
     --nFiles 3\
     --nFilesPerJob 9\
     --outDS user.fatsai.DSNNr_InputsArgTest_v9\
     --nGBPerJob=MAX\
     --nJob 1\
     --nCore 2\
     --notExpandSecDS\
     --notExpandInDS\
     --outputs myOutput.tar.gz\
```