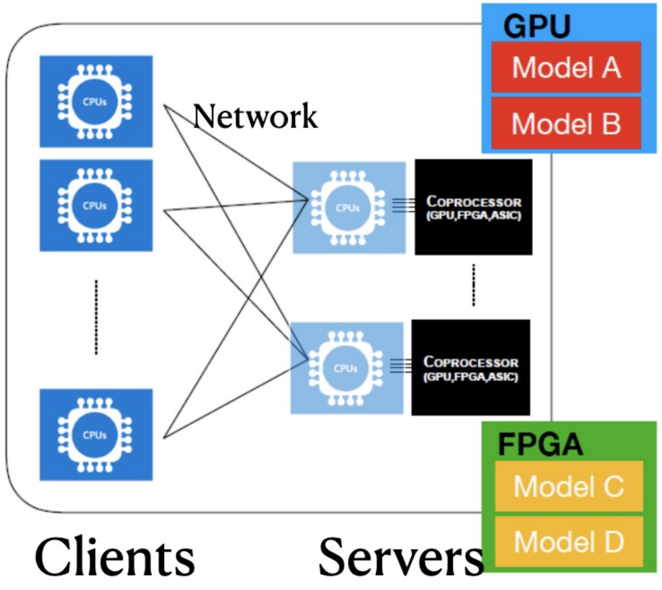
Coprocessors for ML Algorithm Inference: SONIC Features and Performance Updates

Patrick McCormack, Philip Harris, Jeffrey Krupa, Dylan Rankin, Simon Rothman (MIT)
Maria Acosta Flechas, Yongbin Feng, Burt Holzman, Kevin Pedro, Nhan Tran (FNAL)
Miaoyuan Liu, Stefan Piperov (Purdue)
Javier Duarte, Raghav Kansal, Nirmal Thomas (UCSD)



IML Monthly Meeting

July 5, 2022

Context

Traditional Inference

- ML algorithm inference rarely optimized on CPU
- Simple approach = get a computer with a GPU/FPGA/ASIC and perform “direct” inference
  - 1-to-1 correspondence of CPU to coprocessor
  - “Problems”:
    - **Limited** (often non-optimal) usage of co-processors
    - Can also be **expensive**

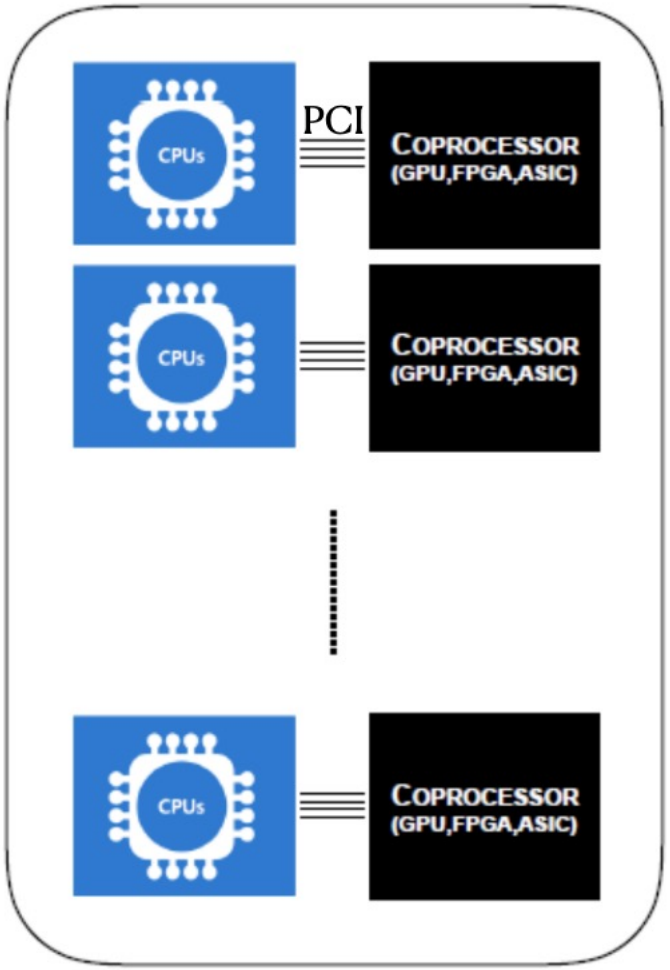
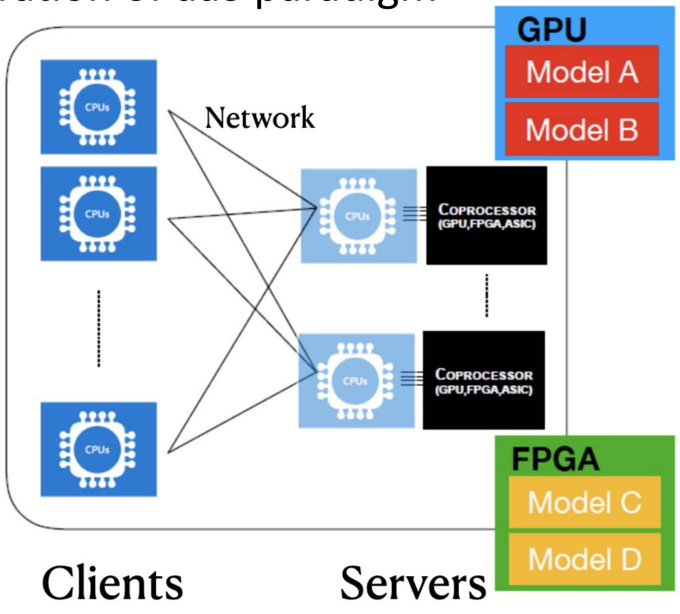


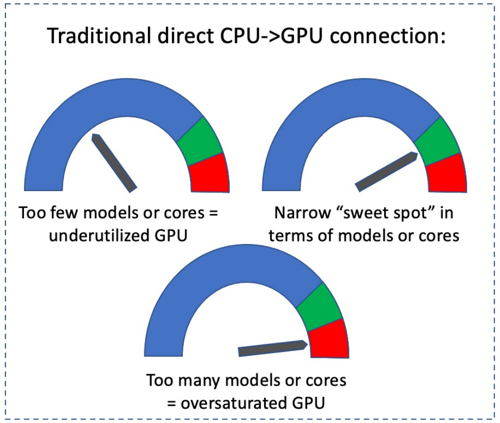
Illustration of “direct” co-processor paradigm

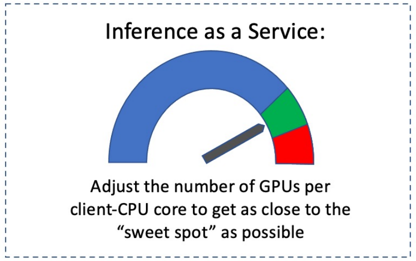
Inference as a Service (aaS)

- Alternative: treat co-processor as distinct server
- Here, CPUs act as clients, preprocessing data for inference and making calls to server
- **Advantages:**
  - CMSSW no longer needs to handle ML framework, just preprocessing and I/O (*can use otherwise unsupported frameworks*, like PyTorch)
  - Take advantage of industry efforts; simple support available for different co-processors with no need to rewrite models
  - One co-processor can service many CPUs – *optimize computational load*
  - Server can provide access to multiple types of co-processor – choose best one on per-model basis
  - Can access remote GPUs (only way to do this)!
  - ***Portable, Simple, Containerizable, and Flexible***

Illustration of aaS paradigm



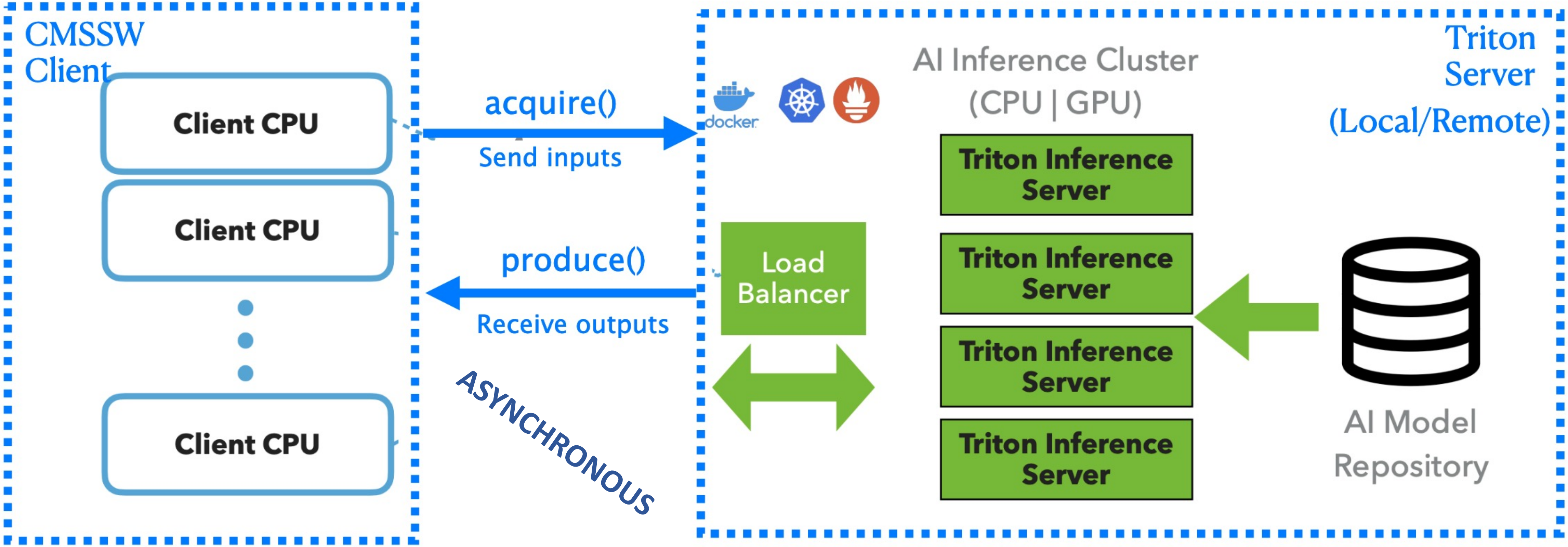




Taking advantage of industry efforts

- **For GPUS**, we can use the NVIDIA Triton Inference Server (more documentation & github)
  - Supports Tensorflow, TensorRT, ONNX, and PyTorch, XGBoost, Scikit Learn (BDT) models + Custom backends
- **In CMSSW**: runs through SONIC (Services for Optimized Network Inference on Coprocessors)
  - Main code in SonicCore and SonicTriton
  - Can make **asynchronous inferences requests**
  - If GPU resource not available, automatically spins up **fallback server on CPU**

SONIC Visualized



- Use acquire() and produce() functions to send/receive information to/from server
  - See example of use for DeepMET algorithm (and model file PRs)

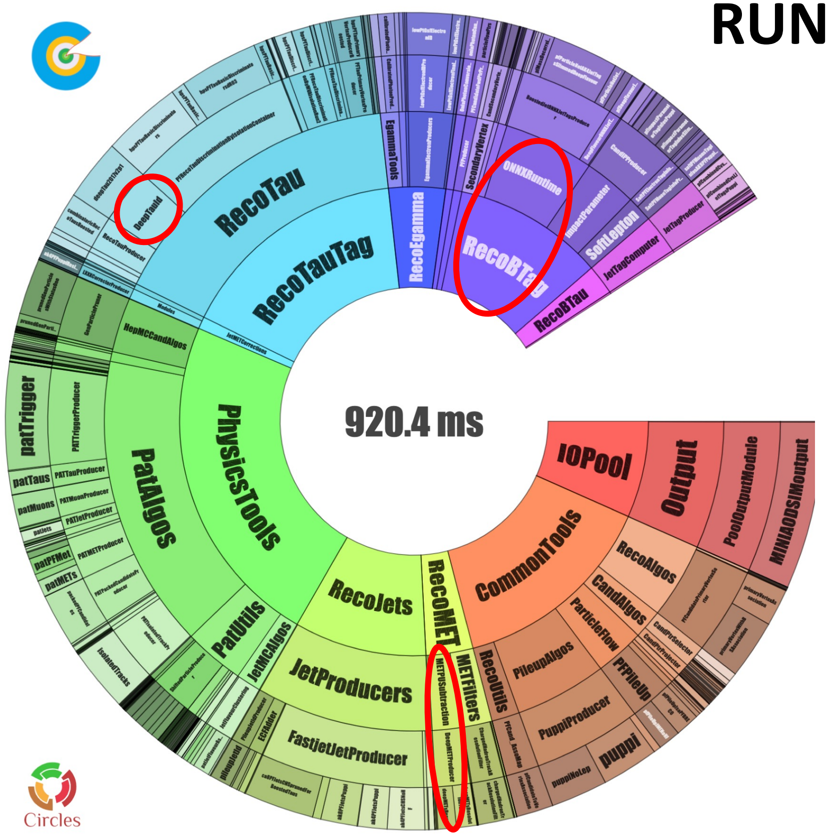
Workflow Demonstrator
– or –
The Benefits of SONIC

MiniAOD Demonstrator

- We made a MiniAOD demonstrator includes SONICized versions of the algorithms circled on the right
  - **ParticleNet** – ONNX model for jet tagging (AK4, AK8 flavor, AK8 MassRegression, and mass) decorrelator
    - See presentation and paper
  - **DeepMET** – TF MET calculation
    - See internal twiki, Yongbin's thesis
  - **DeepTau** – TF model with CNN for tau ID
    - CADI, approval talk, also these older slides
- These account for ~9% of total miniAOD production latency*
  - This was all run on a **ttbar dataset**
- Presentations from the fall:
  - O&C
  - S&C Blueprint

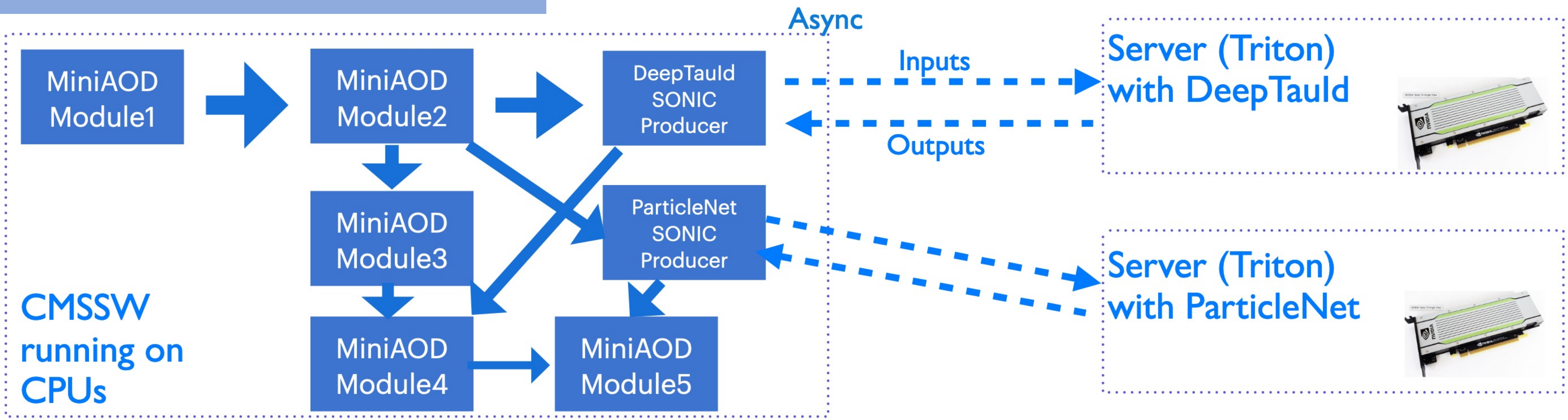
	Time(ms)	Fraction(%)
Total	920.4	100
ParticleNet	43.4	4.7
DeepTau	22.3	2.4
DeepMET	14.0	1.5
Sum	89.7	8.6

RUN 2



*11-12% for Run 3 miniAOD

Explicit Workflow



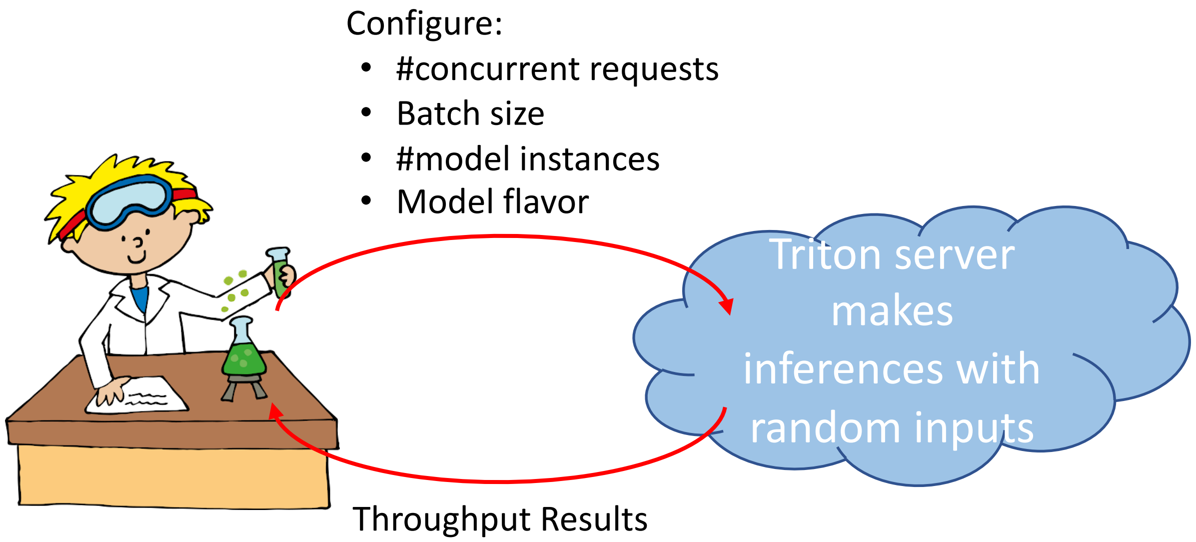
- Servers load the model files (PRs: DeepTau, DeepMET, ParticleNet)
- Three SONIC Producers to do the pre/post-processing and handle the IOs for model inferences (PRs: DeepTau, DeepMET, ParticleNet)
  - We validated object by object that the output between the regular workflow and SONIC producers are identical (up to the numerical precision)

Production testbeds

- Workflow can be deployed in a variety of computing contexts
  - **Google Cloud**: Triton server on cloud VM, with client-side CPUs also in cloud. *Effectively, use cloud as a scalable, temporary tier 2*
  - **Current Tier 2** (Purdue): 2 T4s available – client CPUs at Purdue (can also use cloud GPUs)
  - **HPC computing cluster** (NTU - NCHC): Many GPUs available – client CPUs in Taiwan (can also use cloud GPUs)
- NOTE: Can use CPUs at one site to communicate with GPUs at another site

Optimization: understanding our models

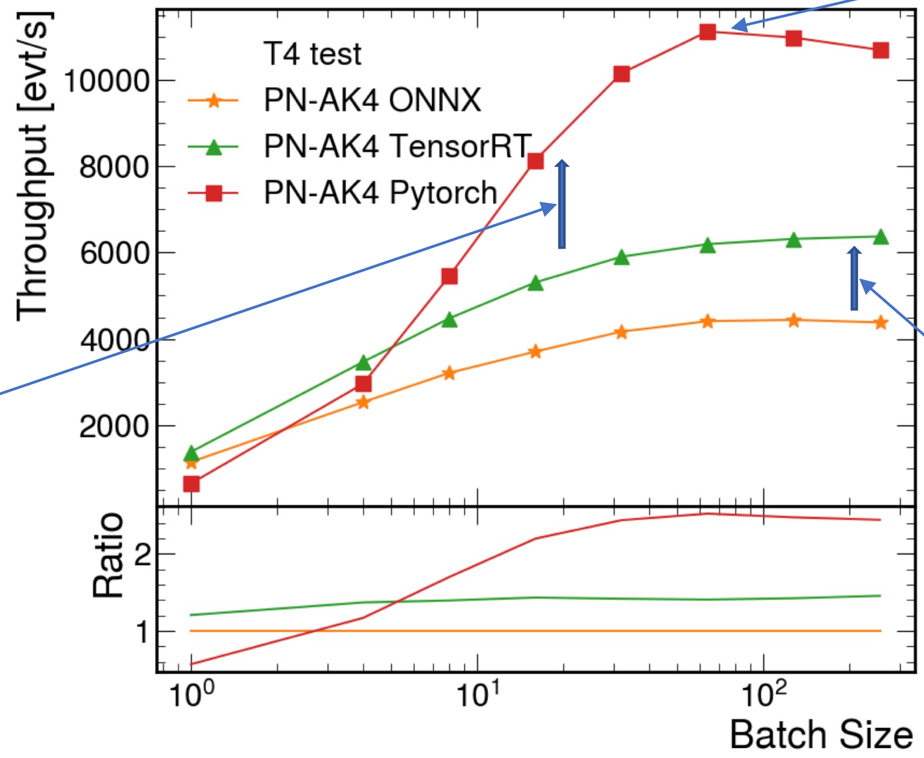
- In the IaaS paradigm, GPU server is wholly separate from client side
  - Triton provides a variety of native **tools to explore model performance**
- Using these tools, we can explore different model configurations to achieve peak throughput



Optimization: perf_client examples

- Tests performed with a Triton server on 1 NVIDIA T4 GPU
  - Single model is loaded into server, and triton’s perf_client feeds in random inputs to determine throughputs
  - Examples of what we can learn:
    - Expected throughputs
    - Optimal configuration
    - “Best” version of model if multiple available

Peak expected performance at batch size ~60 jets



PyTorch preferred at higher batch size (number of jets inferenced at once)

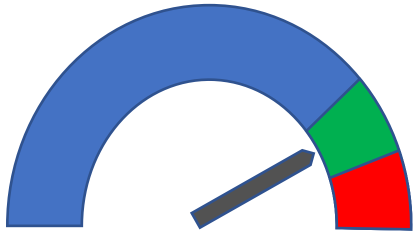
TRT* outperforms bare ONNX

*TRT=TensorRT. An NVIDIA-specific algorithm to optimize performance on NVIDIA GPUs

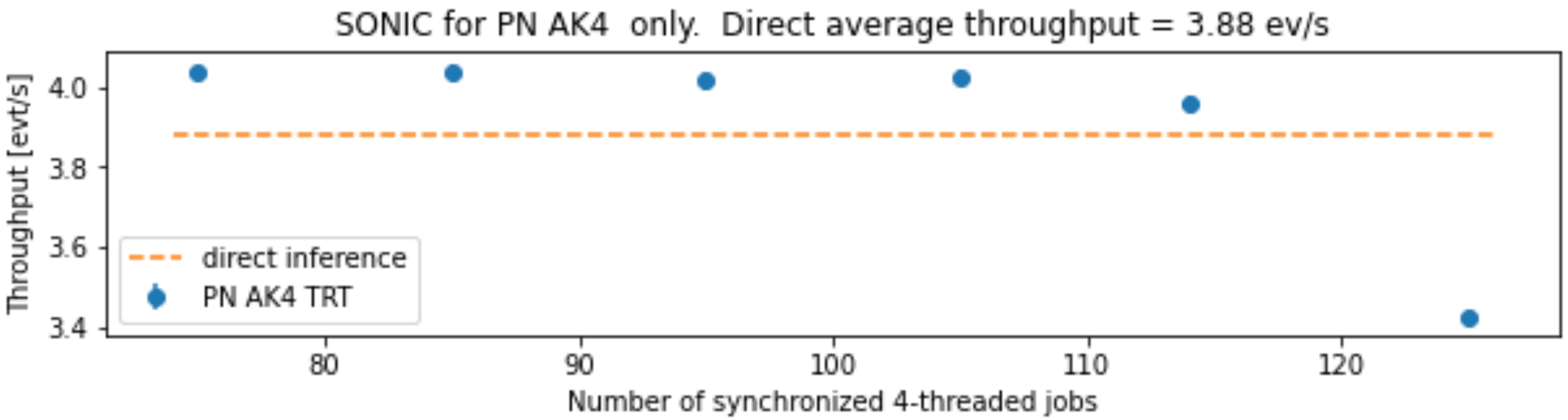
Optimization: saturation scan examples

- We also want to know how many client-side jobs a single GPU can handle
  - Allows us to reach the GPU-per-CPU “sweet spot”
  - Also allows us to calculate how many GPUs we need

Inference as a Service:



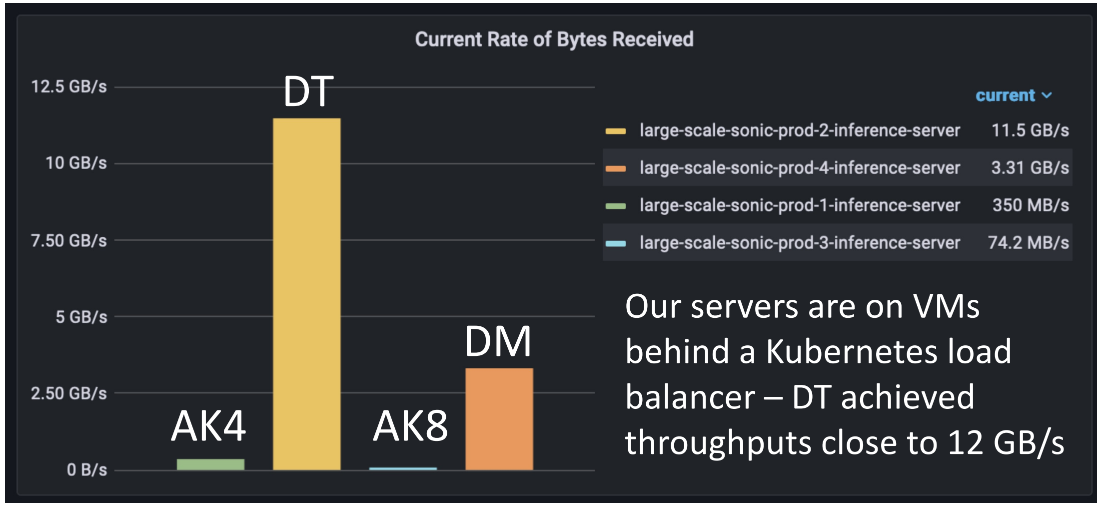
Adjust the number of GPUs per client-CPU core to get as close to the “sweet spot” as possible

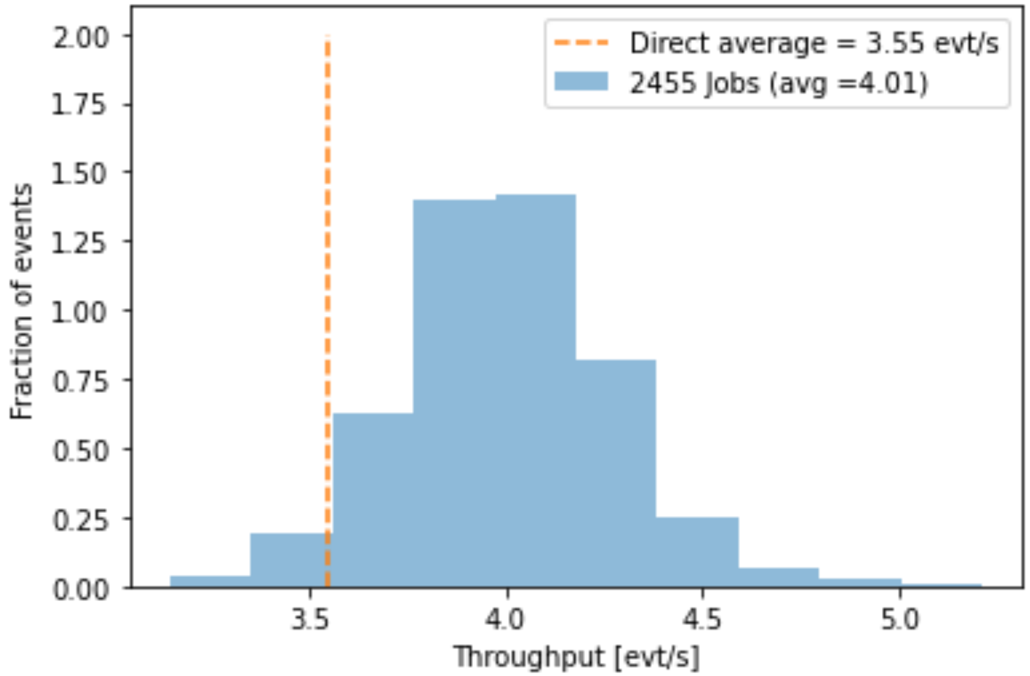


Results for TRT-ized version of PN-AK4. 1 GPU can handle about 115 simultaneous 4-threaded jobs, but for server setup, might want to use ~105 jobs to be safe

Demonstration: Scale tests

- Optimization can be performed with just a few GPUs per algorithm of interest
- For a demonstration of how this scales up, we can deploy ~100 GPUs to service thousands of CPU nodes, all in the cloud

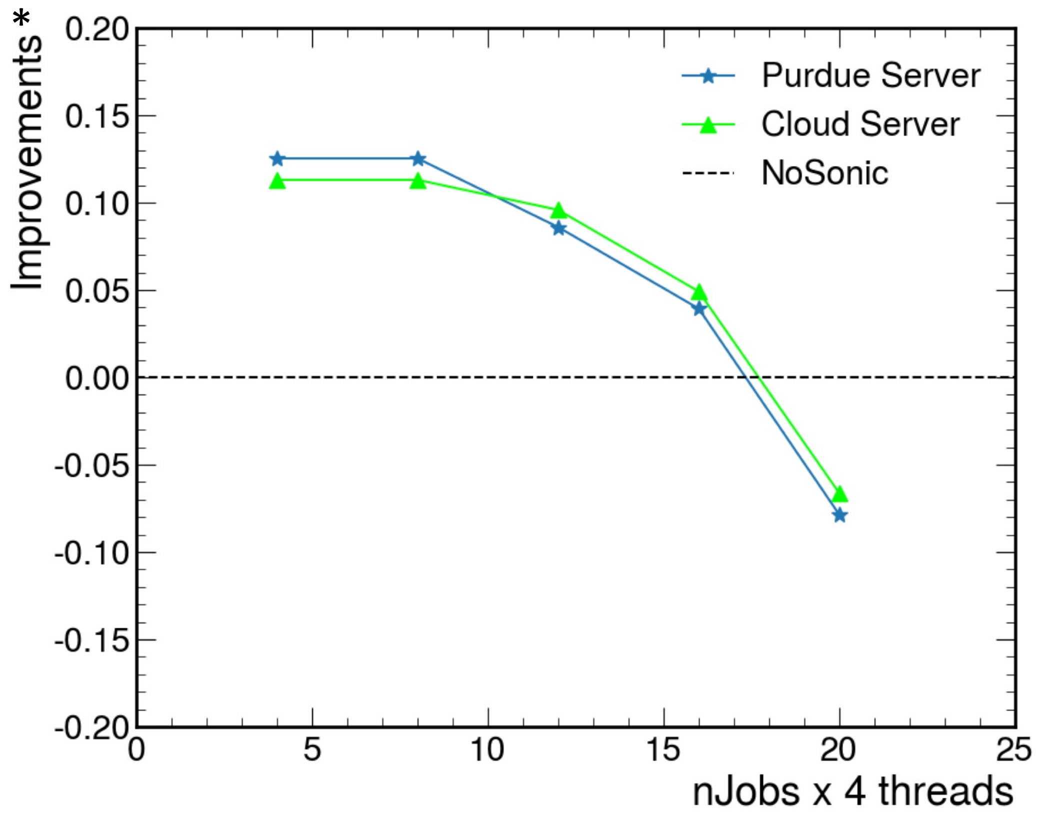




Here, we ran with 10,000 CPU cores and ~100 GPUs, achieving expected speed-up in processing. **NOTE:** with direct inference (say 32-to-1 CPU to GPU ratio), this would take ~300 GPUS

Demonstration: Server location robustness

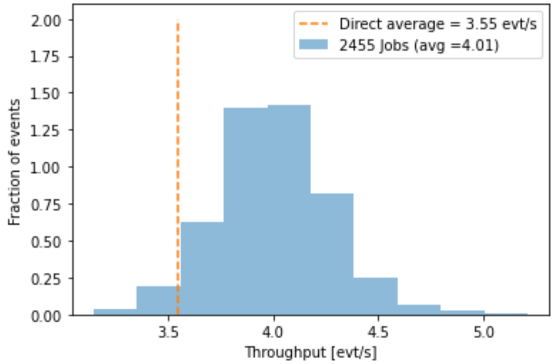
- Comparing the throughputs between one server at Purdue and one server in Google Cloud VM
  - CMSSW jobs running at Purdue T2 (Run-3) workflow
  - Similar results between Purdue server and GCP server

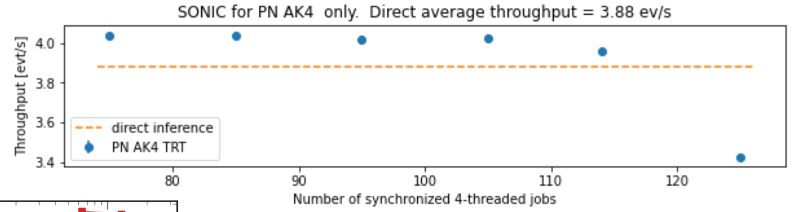


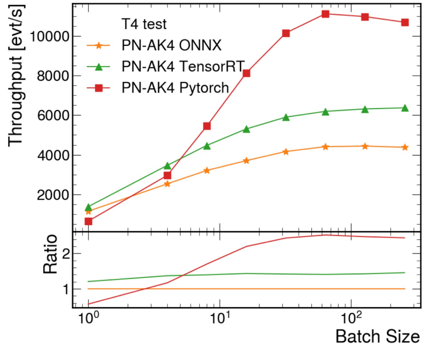
*Improvements = fractional increase in throughput relative to CPU-only inference

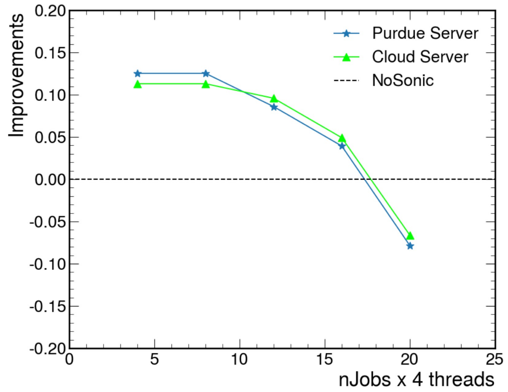
SONIC: Summary of benefits

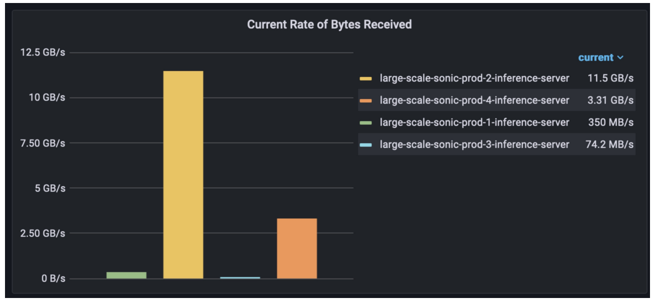
- Why use SONIC?
  - **Increase throughput**
    - GPUs enable acceleration of ML algorithms
  - **Optimize GPU-to-CPU ratio**
    - Save money if looking to buy GPUs or increase utilization of current resources
  - **Flexibility of algorithm design + optimization**
    - Not restricted to currently supported frameworks in CMSSW
    - Can tweak deployment parameters to optimize inference rates and use e.g. TRT
  - **Not restricted to local GPUs**
    - You just need access to a GPU, you don't need to buy one necessarily
  - **Bandwidth limitations not yet seen in realistic deployments**
    - We got up to 12 GB/s of data transfer for deepTau in our scale tests









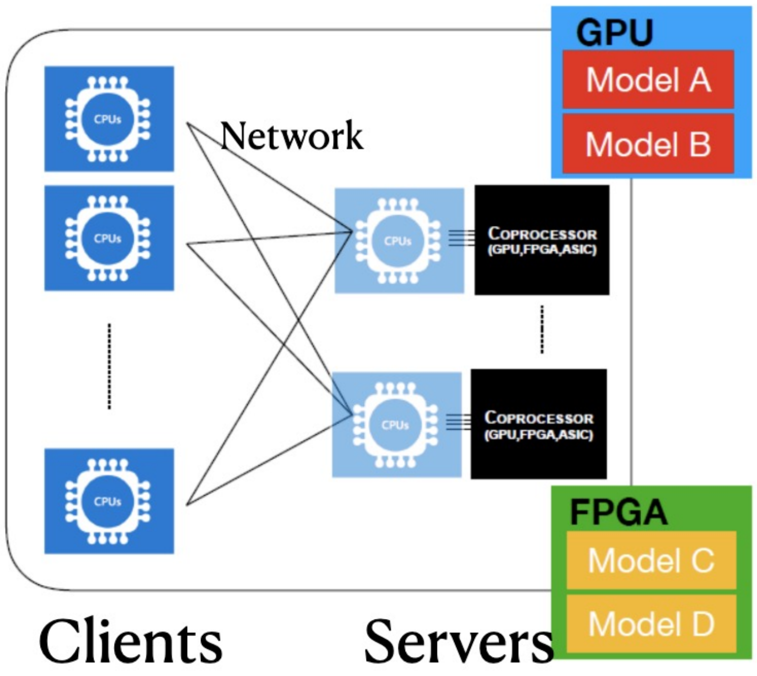


Looking Forward

GPU server deployment

- On our to-do list: create resource allocation framework
  - Assign GPUs to jobs
  - Automated way to spin up GPU-based servers
  - Ensure that we don't saturate GPU resources

Want to make this as easy as possible for users



Additional Projects

- Beyond data processing, SONIC could be **useful for analysis** groups
  - Let's say you have a complex ML algorithm but no GPU
  - Perhaps in the future, you could deploy a server on some collaboration-wide shared GPU resources and run using that
  - "**PySONIC**" framework under development
- Implementing some additional Triton features, such as "**ragged batching**"
  - E.g. Currently have to zero-pad tensors for ParticleNet, increasing network demands and latency
  - "Ragged batching" makes zero-padding unnecessary
- New demonstrator algorithms under development (only possible with SONIC for now)
  - **ECAL Dynamic Reduction Network** regression – PyTorch Geometric based algorithm [S. Rothman]
  - **SPVCNN** algorithm to use depth information in HCAL clusterizing
- Algorithms in the pipeline
  - MLPF
  - Tracking as a service (ExaTrk or ACTS)

Summary

- Coprocessors, such as GPUs are becoming increasingly important
  - But they can be expensive and not widely available
- SONIC is a powerful tool to optimize coprocessor utilization and availability
  - Already demonstrated to deliver expected throughput increases at large scale for MiniAOD reprocessing
- Physics case for SONIC: we can deploy models in any format supported by Triton and even make custom backends
  - More flexibility than what is currently in CMSSW!
- Working on robustness studies and developing ease-of-use tools

Backup

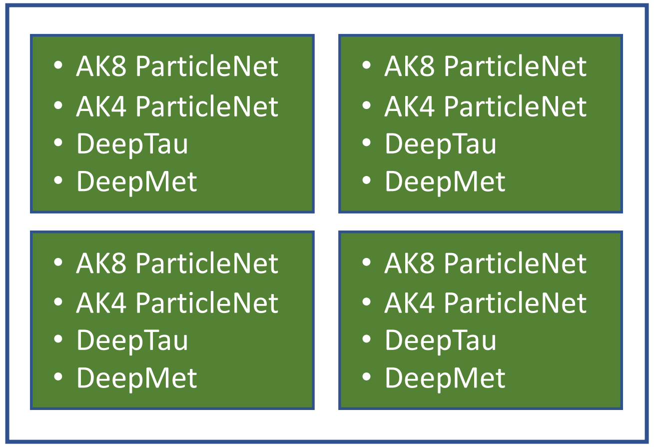
SONIC: Some Current Features

- Currently available:
  - **TritonService**: tracks available servers and models
  - **Local fallback server**: will automatically start a server on local GPU or on CPU if model not on remote server
  - **Shared memory**: can speed up inference on local servers – for CPU uses memory-backed temporary file system and for GPU uses CUDA to copy directly to GPU memory
  - **I/O compression**: Use some CPU resources to compress I/O to use less bandwidth
  - **SSL Authentication support**
  - CMSSW ProcessModifier "**enableSonicTriton**" turns on SONIC features
    - Also "**allSonicTriton**" to enable full workflow

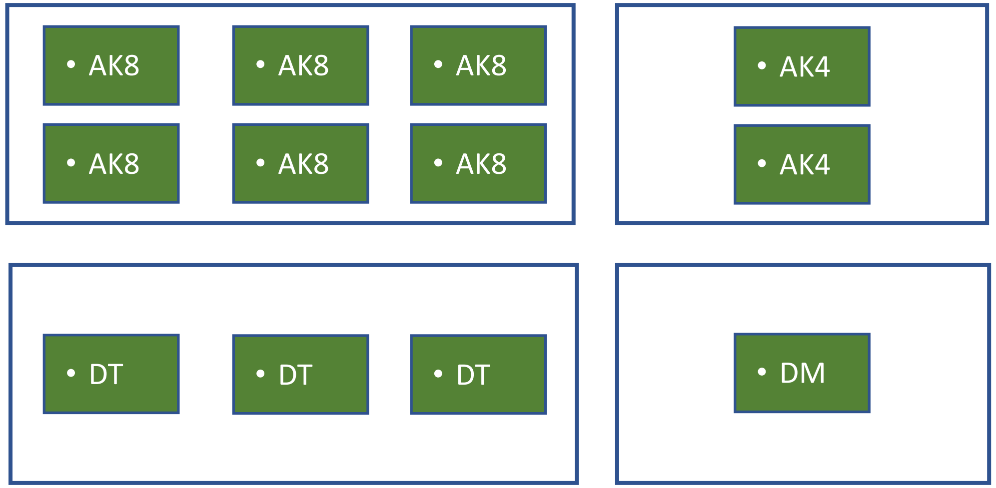
- Still to come:
  - **Client-side ragged batching** – use of different size inputs vectors within the same batch; in contact with NVIDIA now (3x speed up when ragged batching is used in direct CPU inference for ParticleNet jet tagging)

Optimization: server deployment

- We can **choose how to distribute our models** over GPUs
  - Difficult option to probe in perf_client
- Main options:
  - Load every model onto every GPU
  - Load models onto separate GPUs and run with multiple servers
  - Use different combinations of models on GPUs
- In practice, we often see slightly better performance with split models
  - ~3% fewer GPUs needed
  - This is likely sample dependent (most R&D here used ttbar samples)



One IP address

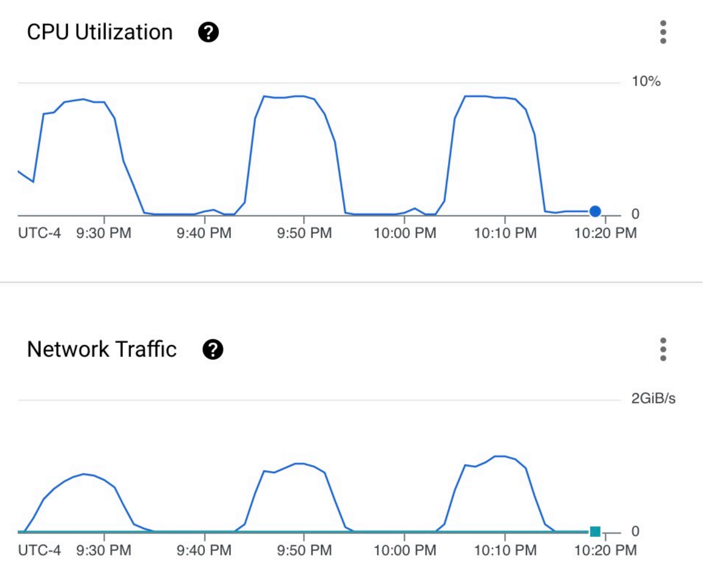


4 IP addresses in this example

Note: How many server-side CPUs do we need?

- Disclaimer: in our Google Cloud VMs, we give each GPU 24 CPU cores
  - **HOWEVER:** this is because allowed bandwidth in Google Cloud is somewhat restricted based on the number of CPUs in a VM
- Triton doesn't actually use all of these cores
  - We can monitor the actual CPU utilization at the saturation point, as below

Model	CPU utilization percent (24 cores available)	Cores used per GPU
PN AK4 TRT	4.8%	1.15
PN AK8 TRT (all 3 models)	8.4%	2.02
DeepTau TRT	8.93%	2.14
DeepMET	17.48%	4.19
ALL Models on 1 GPU (saturation at ~32 4-threaded jobs)*	16.09%	3.86



76 jobs/GPU 90 jobs/GPU 100 jobs/GPU

Example of bandwidth and CPU utilization monitoring in 2-GPU VM servicing DeepTau for runs with various numbers of synchronized jobs

***In this simplified scenario, 128 client-side cores are serviced by 1 GPU with 4 server-side CPUs**