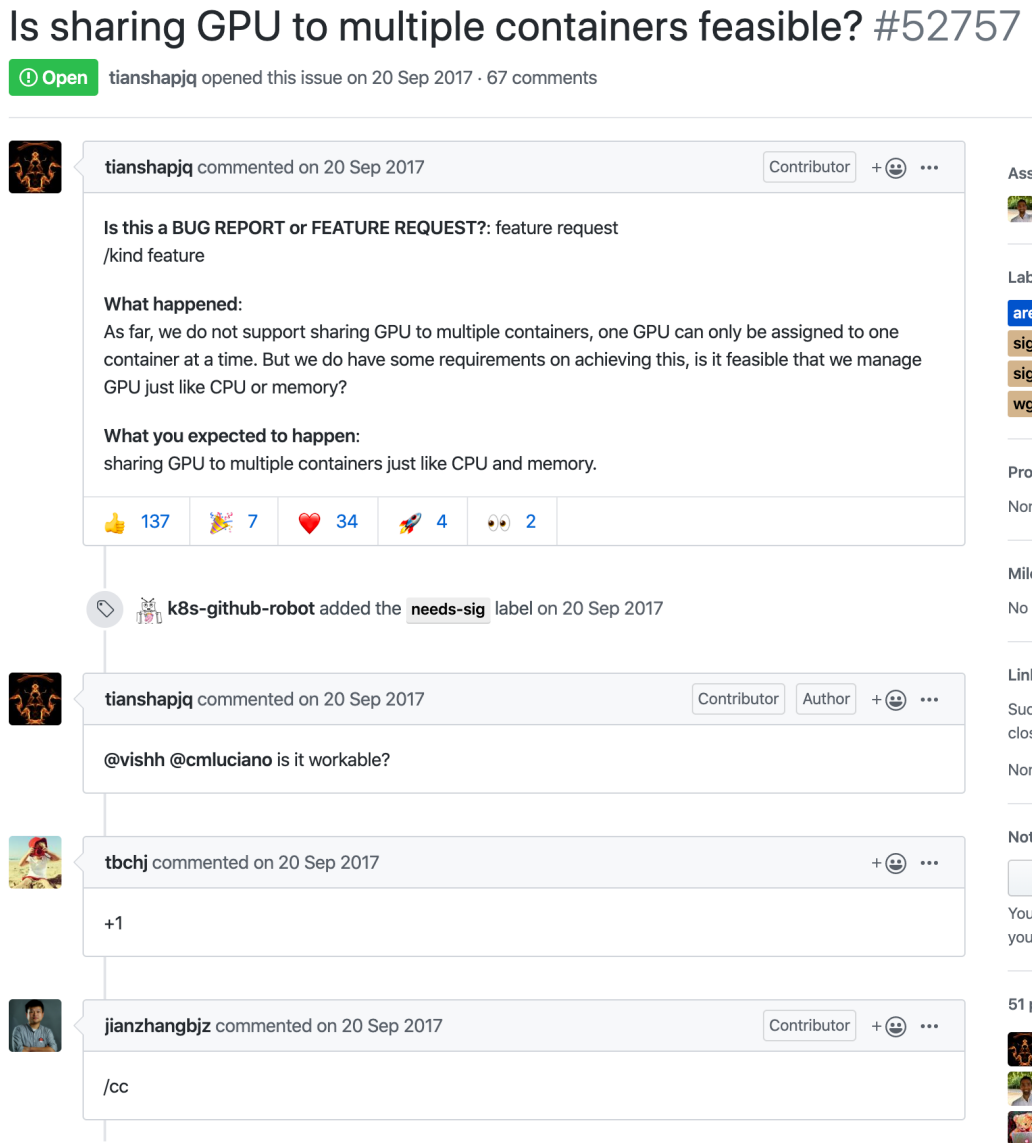
Is Sharing GPU to Multiple Containers feasible?

Samed Güner, SAP Artificial Intelligence
August 18, 2020



THE BEST RUN SAP

About me

- Developer at SAP Artificial Intelligence
- Contributor to Terraform and Cloud Foundry
- Living in Germany, having Turkish roots

Open Source Projects

Contributions and Commitments from SAP

CLOUD NATIVE COMPUTING FOUNDATION
Platinum Member

ECLIPSE FOUNDATION
Strategic Member

CLOUD FOUNDRY
Platinum Member

kubernetes

Gardener

Kyma

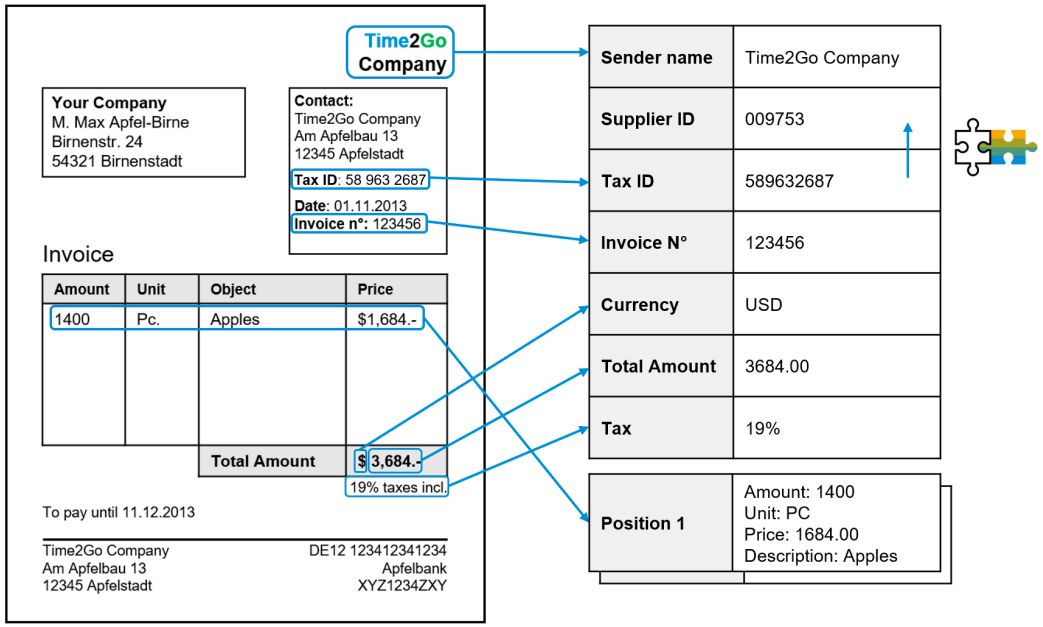
Luigi

Sap Machine

...

Artificial Intelligence at SAP

- Providing AI platform and services for enterprises and SAP applications
- Embedding intelligence into enterprises
- Large number of models in production



AI Service: Document Information Extraction

ML on GPUs in K8s

- Expensive workload
- GPU utilization as challenge
- Sparse and limited availability

Fictional Calculation:

$u = 70\% = 0{,}7 \quad n = 500 \quad p = \$3{,}06\ \frac{1}{h}$ [1]

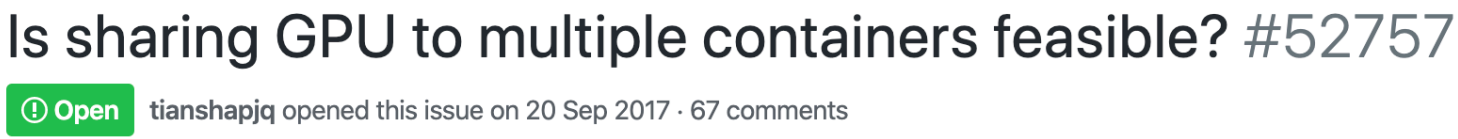
$C(n) = (1 - u) * n * p$

$= \left(0{,}3\ * 500\ * \$3{,}06 \frac{1}{h}\right) * 768\ h$

$= \$352.512\ loss\ per\ month$

[1] averaged price from AWS, Azure and GCP, smallest setup with one Nvidia Tesla V100

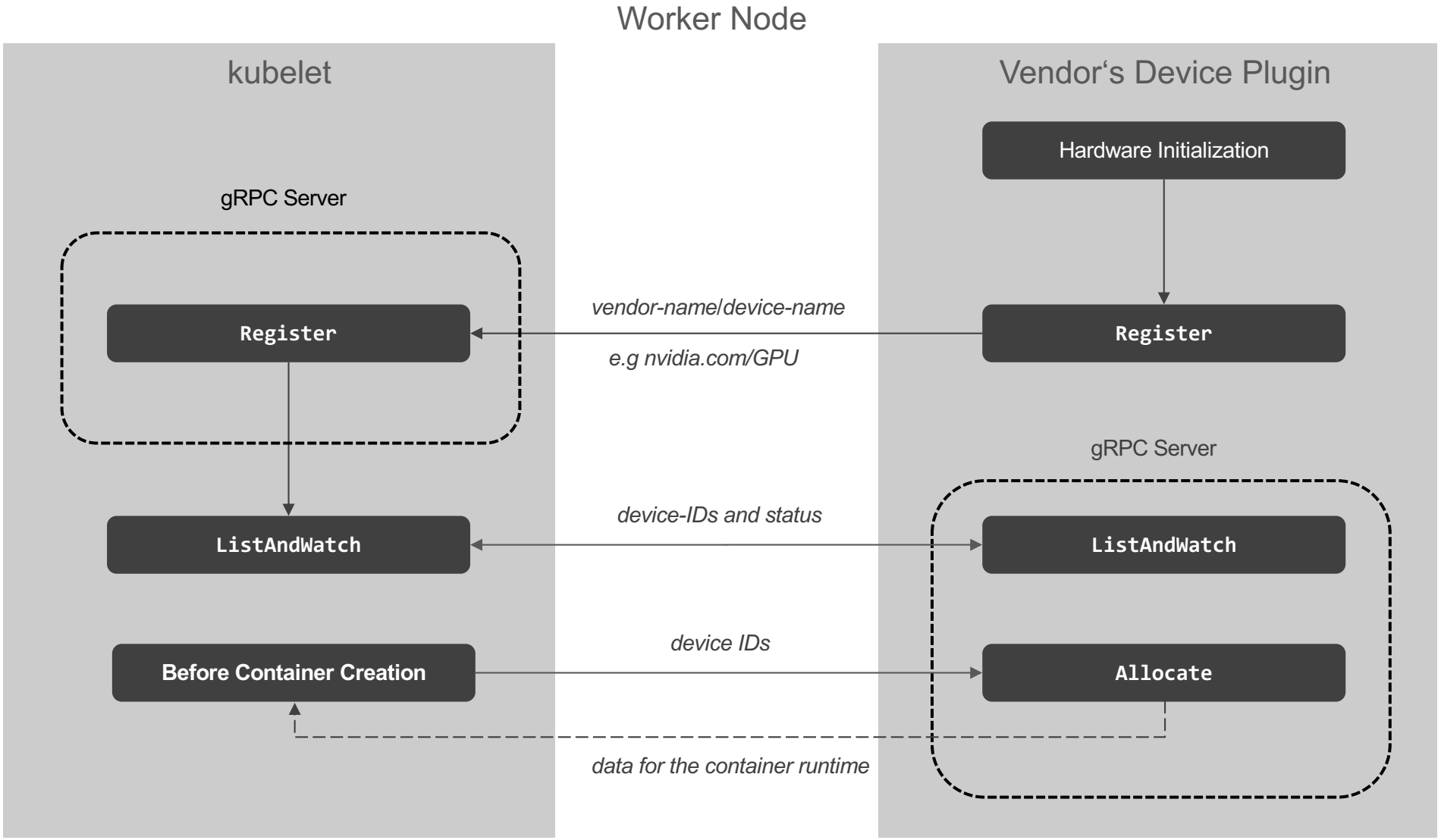
So we asked ourselves ..



.. too[1].

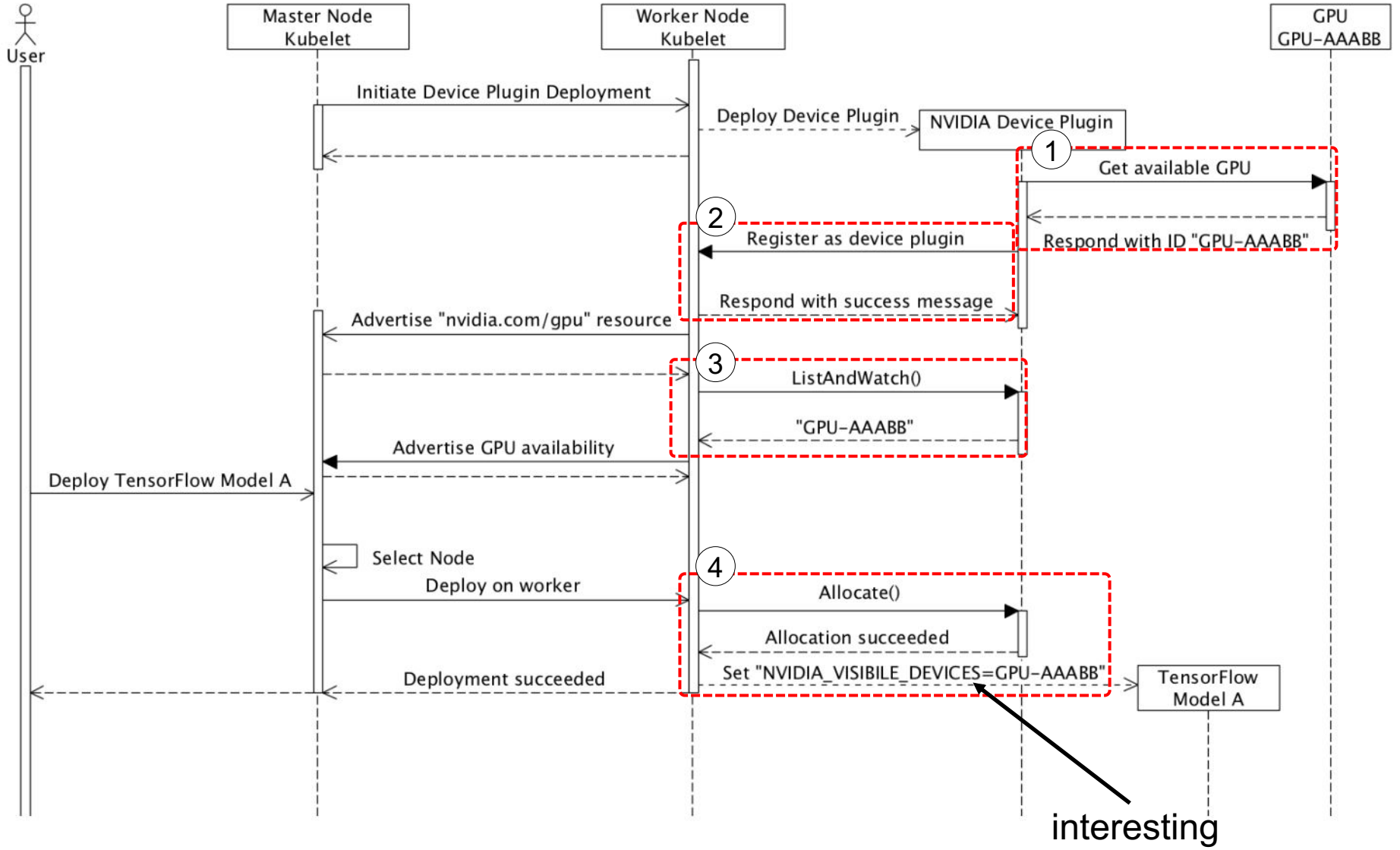
[1] target: Nvidia GPUs

K8s Device Plugins[1]



[1] as of K8s 1.10

So how are GPUs provisioned in K8s?[1]



1 **INIT**

2 **REGISTER**

3 **LISTANDWATCH**

4 **ALLOCATE**

[1]this applies only for Nvidia GPUs

So how did we share a GPU?

- Variant 1: Model Stuffing in Application Logic
  - no CGROUPS for cpu/memory
  - no ability to do request rate limiting per model

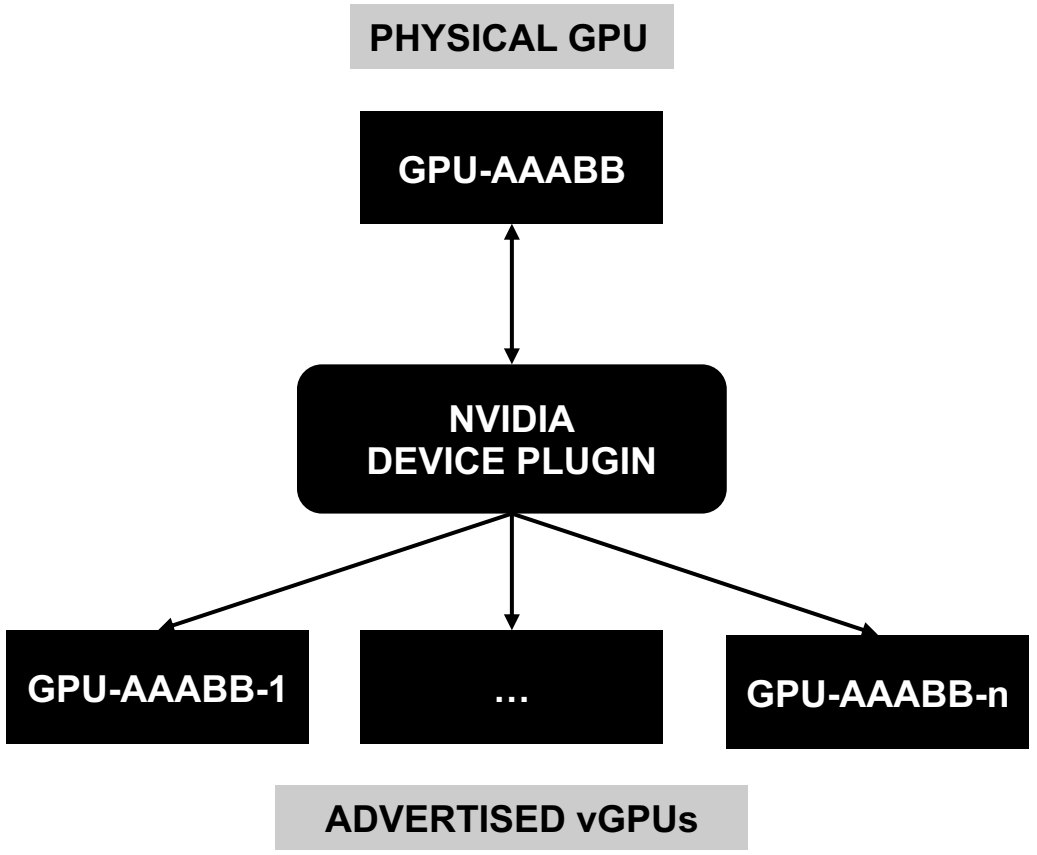
- Variant 2: Advertising more GPUs to K8s
  - extending the Nvidia Device Plugin for vGPUs

- Variant 3: NodeSelector Hacking
  - Privileged Container consume GPU directly
  - No Kubernetes Scheduling at all

So how do we advertise more GPUs?

problems already start here

```
1   for k := 0; k < NUM_VGPU; k++ {
2       // generate vGPU ID
3       vGPUId := GPUtovGPU(d.UUID, k)
4       // generate the vGPU
5       tmpVGPU := vGPU{vGPUId, 0, 0, 0,
6           pluginapi.Healthy}
7
8       // assign the vGPU to the physical GPU
9       gpus[d.UUID].vGPUs[vGPUId] = tmpVGPU
10      // make it public in ListAndWatch
11      devs = append(devs, &pluginapi.Device{
12          ID: tmpVGPU.UUID,
13          Health: pluginapi.Healthy,
14      })
15  }
```



We asked ourselves ..

- How many models can we fit on a Nvidia K80s?
- How does the whole system behave?
- What are the trade-offs in doing so?

.. so we did our experiments and collected data.

Collecting Data from NVIDIA Device Plugin

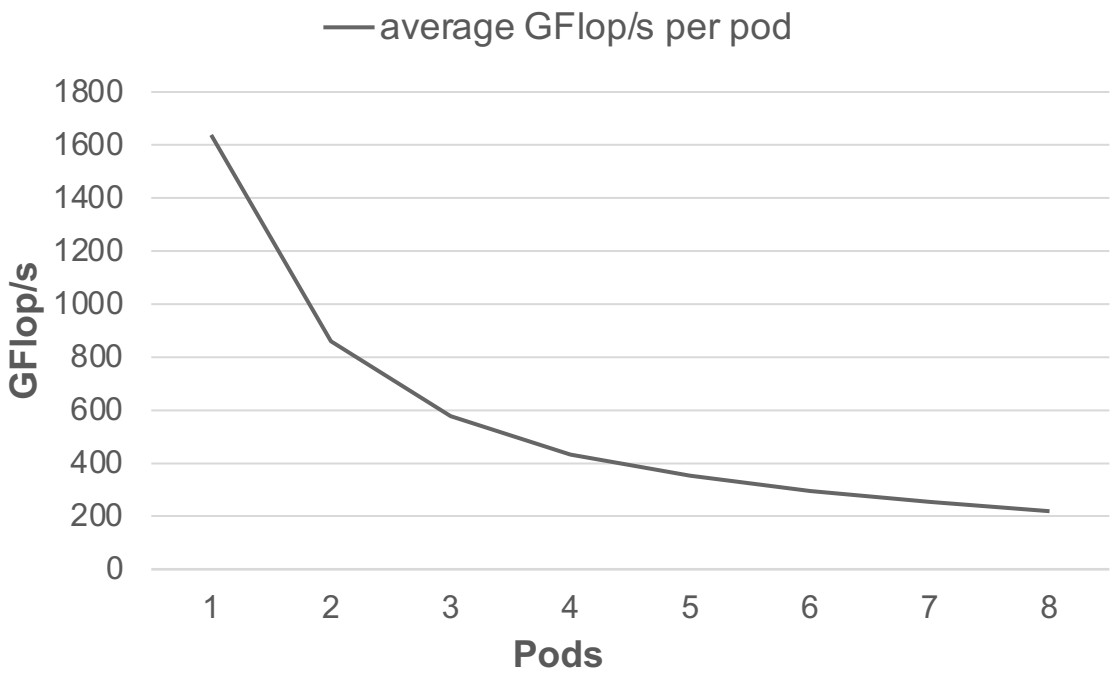
- Device Plugins are K8s Resources!

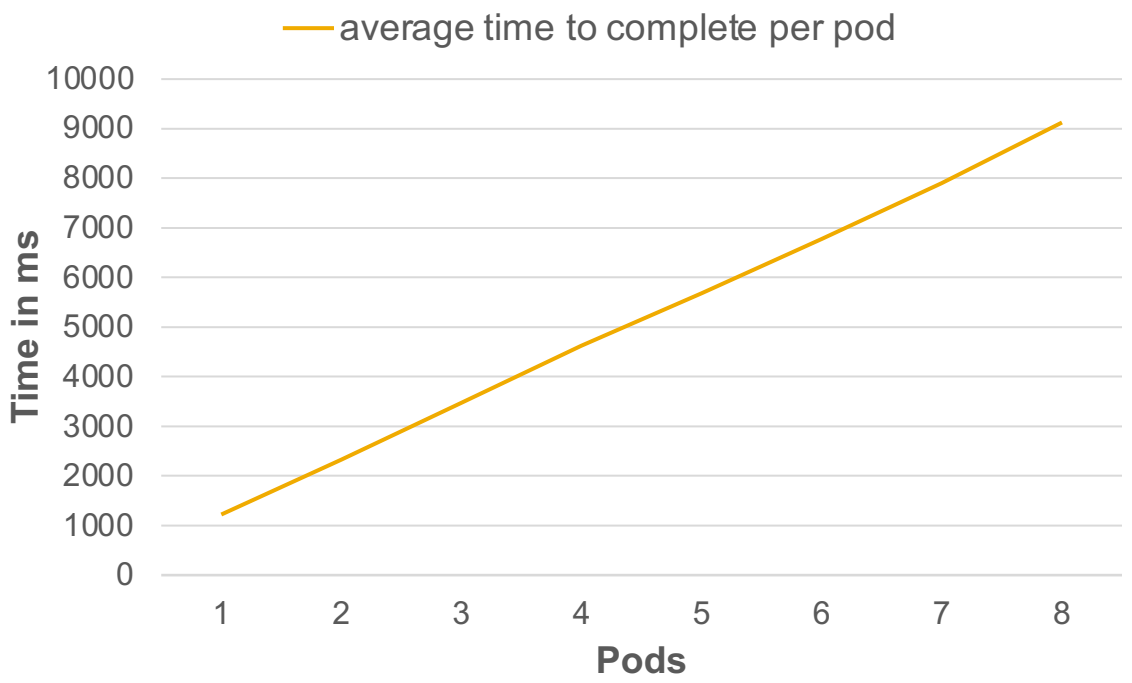
- Collect Data with Nvidia Management Library (NVML)
  - Low Level Access to GPU
  - Go-C Bindings
  - Used by `nvidia-smi`

- Expose via Prometheus and Grafana

Experiments[1]

nbody Simulation (n= 100096)



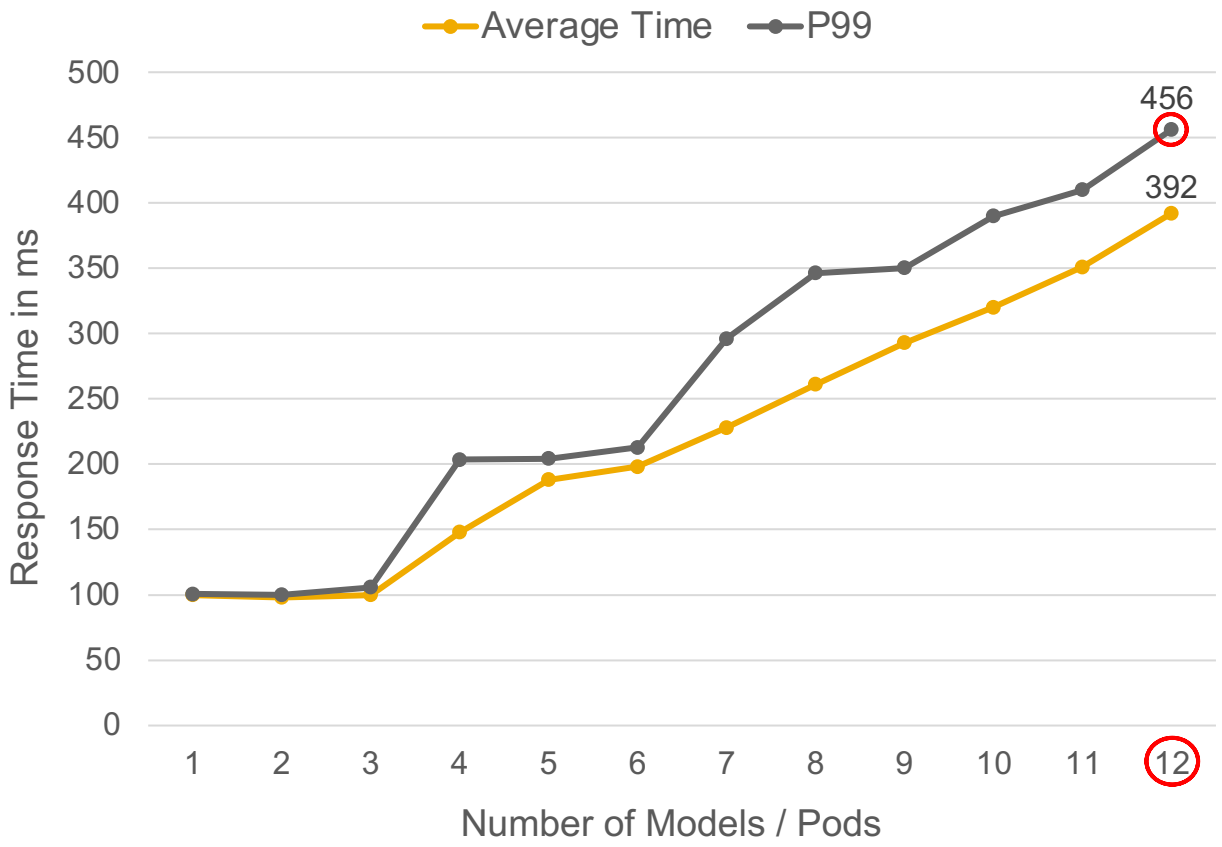


- Missing aspects for ML inference (but comparable to ML training)
- Fair-Scheduling (?)
- No GPU RAM Limiting & Fair Share of CPU

[1] for all experiments we use p2.xlarge with 1x Nvidia Tesla K80 (12 GB)

Experiments[1]

ML Inference InceptionV3 (sequential requests)



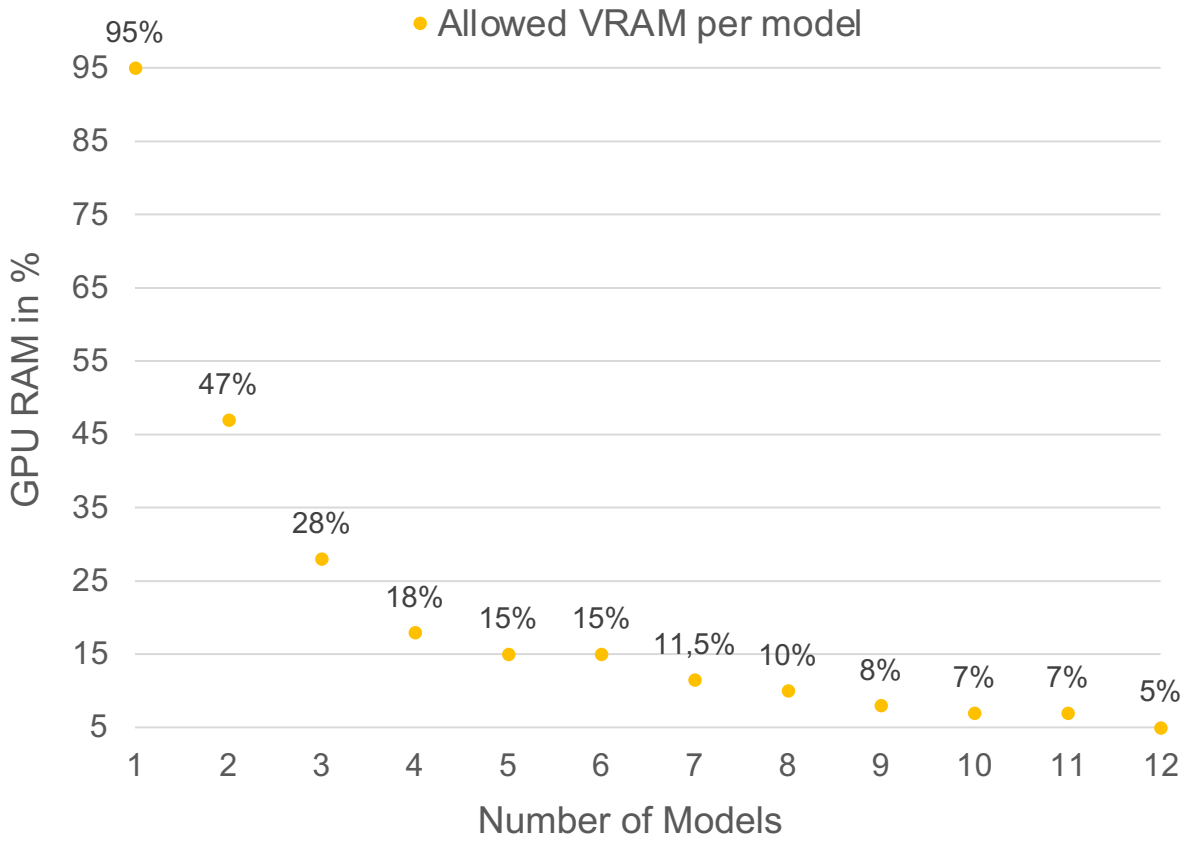
- Low Throughput: 1 Request per Model
- 10,000 requests per Pod
- CPU Limit: 350m per Pod

[1] for all experiments we use p2.xlarge with 1x Nvidia Tesla K80 (12 GB)

Experiments[1]

Dealing with GPU RAM for InceptionV3[2]

- Limiting Tensorflow Serving (TFS) Ram:
  - `tf.GPUOptions`
- Dynamic Scheduling:
  - Offer vGPUs until memory is full
- Can we go deeper?

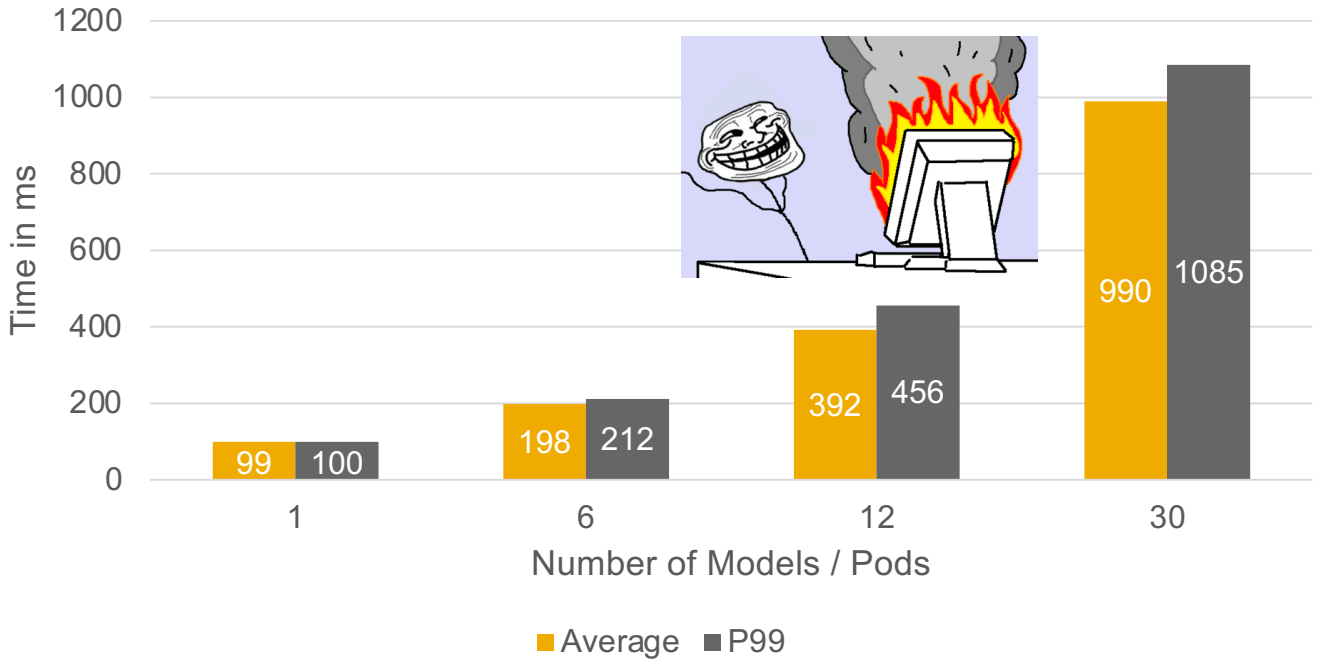


[1] for all experiments we use p2.xlarge with 1x Nvidia Tesla K80 (12 GB)

[2] only applicable for TensorFlow 1.x

Experiments[1]

Dealing with GPU RAM for InceptionV3[2] (sequential requests)



- Minimum 228 mb per Model
  - up to 50 models on one K80
  - TFS will crash for less!
- 30 Models with each having:
  - 100m CPU Limit
  - 3% of VRAM
  - 1 Request at time

[1] for all experiments we use p2.xlarge with 1x Nvidia Tesla K80 (12 GB)

[2] only applicable for TensorFlow 1.x

Experiments[1]

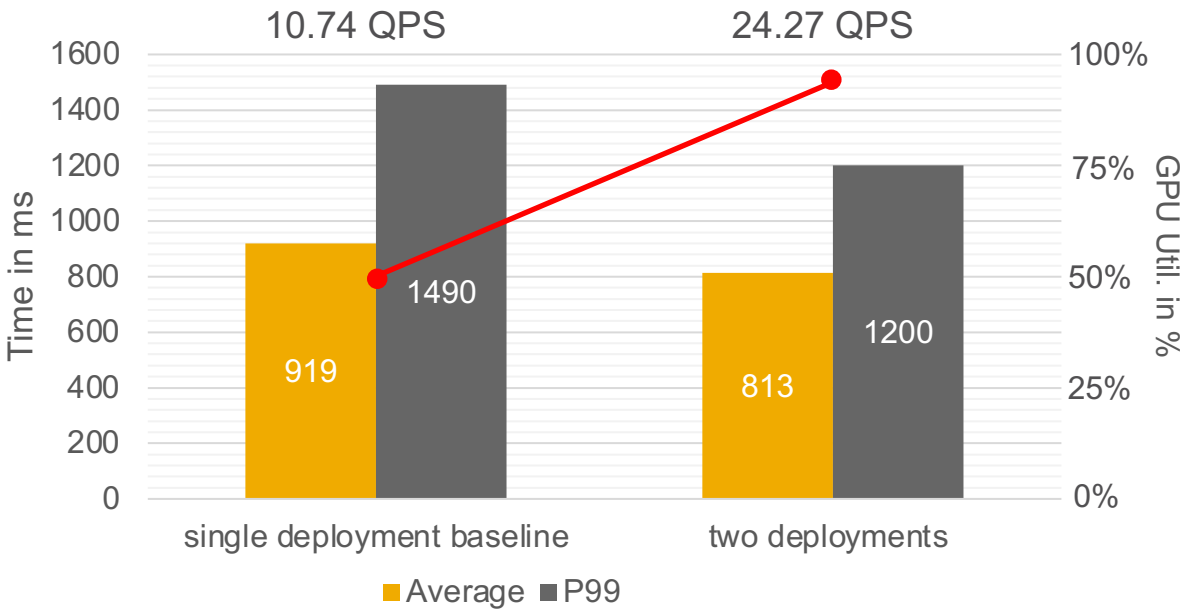
ML Inference InceptionV3 (parallel requests, no batching)

- Limits:
  - 10 parallel requests per Model
  - 350m CPU Limit
  - 3% of VRAM

- 4x worse than sequential request pattern

- Increase in throughput leads to increase in latency and GPU util.

⟶ GPU has more work to do!



With batching enabled 2x latency and throughput improvement for 6.6x more VRAM.

[1] for all experiments we use p2.xlarge with 1x Nvidia Tesla K80 (12 GB)

[2] only applicable for TensorFlow 1.x

Germans would say ..

Jein[1]

(yes, but with trade-offs and limitations)

[1]an answer between yes and no

What does that all mean?[1]

ML Inference

- Our solution is far away from production-ready

- GPU sharing is possible, if you try hard.
  - up to 30x cost saving[2]
  - Trade-Off between Throughput and Latency

- CPU limits as key factor in our setup
  - can be used to in-/decrease GPU Utilization, thus latency and throughput
  - allows deployment of multiple models with similiar latencies while doubling throughput
  - GPU Util. overcommitment will increase latency

[1] for all experiments we use p2.xlarge with 1x Nvidia Tesla K80 (12 GB)

[2] K80 in combination with InveptionV3 capped at 228 MB VRAM and 100m CPU

Limitations with our GPU Sharing

- No isolation gurantee on hardware level by Nvidia[1]
  - Multi-Tenancy setup is impossible
  - Still some sort of fair-scheduling of GPU duties

- No Low-Level API to specify GPU and VRAM
  - CPU Limits and TFS the only current way[2]

- “Scheduling” only by offering vGPUs from Device Plugin

- Resource Fragmentation

[1] for Nvidia Tesla K80

[2] without major customizations on CUDA level

What do we need for GPU Sharing?

- Scheduling / isolation on GPU-level
  - CGROUP-like isolation of GPU resources
- Resource Defragmentation / Locality-Awareness
- Low-overhead during scheduling and processing
- Per device sub-resources (GPU Util. and VRAM) exposed at device plugin

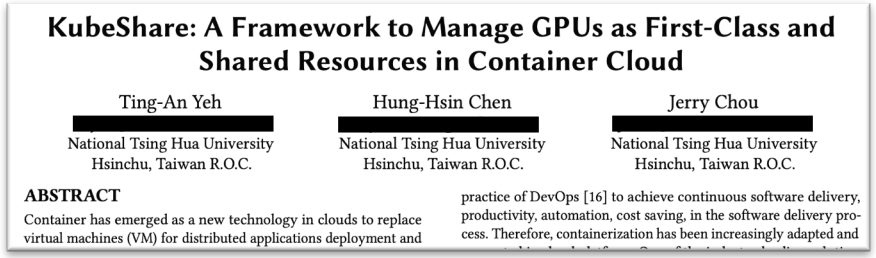
What does the community do?

- Deepomatic (github.com/Deepomatic/shared-gpu-nvidia-k8s-device-plugin)
  - Multiple vGPUs per GPU (cf. our implementation)

- Tencent's GPU-Manager (https://github.com/tkestack/gpu-manager)
  - Cuda-Core / VRAM Requests & Limits
  - Custom Wrapper around Nvidia Device Library

```
apiVersion: v1
kind: Pod
metadata:
  name: vcuda
  annotations:
    tencent.com/vcuda-core-limit: 50
spec:
  restartPolicy: Never
  containers:
  - image: <test-image>
    name: nvidia
    command:
    - /usr/local/nvidia/bin/nvidia-smi
    - pmon
    - -d
    - 10
    resources:
      requests:
        tencent.com/vcuda-core: 50
        tencent.com/vcuda-memory: 30
      limits:
        tencent.com/vcuda-core: 50
        tencent.com/vcuda-memory: 30
```

- KubeShare (github.com/NTHU-LSALAB/KubeShare)
  - Released June 2020
  - Set of custom controller / CRs
  - Sharing through CUDA Calls API interception
  - Powerful: Fine-grained allocation, isolation, locality-aware, low overhead



KubeShare: A Framework to Manage GPUs as First-Class and Shared Resources in Container Cloud

Ting-An Yeh [redacted]
National Tsing Hua University
Hsinchu, Taiwan R.O.C.

Hung-Hsin Chen
National Tsing Hua University
Hsinchu, Taiwan R.O.C.

Jerry Chou [redacted]
National Tsing Hua University
Hsinchu, Taiwan R.O.C.

ABSTRACT

Container has emerged as a new technology in clouds to replace virtual machines (VM) for distributed applications deployment and practice of DevOps [16] to achieve continuous software delivery, productivity, automation, cost saving, in the software delivery process. Therefore, containerization has been increasingly adapted and

Thank you.

Contact information:

samed.guener@sap.com

 / samedguener