CLOUD NATIVE & KUBERNETES AI DAY NORTH AMERICA

Incremental GPU Slicing in Action

Abhishek Malvankar, Olivier Tardieu
IBM Research

Goals

- What?
  - Reduce the cost of running AI workloads on Kubernetes
- How?
  - Dynamically pack multiple AI workloads on the same GPU

Enable incremental GPU slicing

Today's talk

Right-size workloads

https://sched.co/1i7oh

Outline

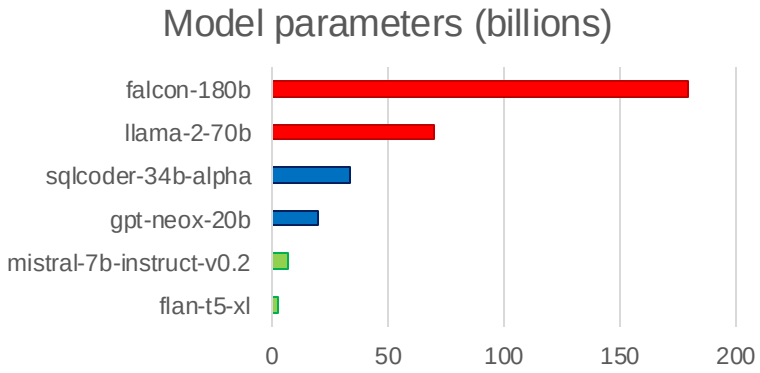
- Motivation
- Multi-Instance GPU (MIG)
- MIG support in Kubernetes
- InstaSlice Operator
- Batch GPU workloads: Quotas, Kueue, and InstaSlice

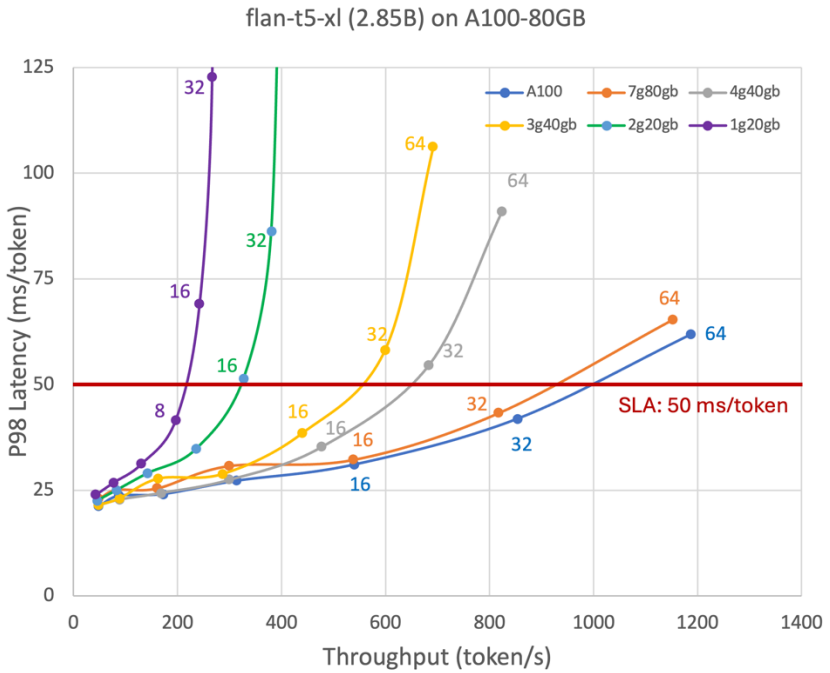
Motivation

- Serving Large Language Models
  - compute/memory requirements depend on model ***and load***

  ⇒ packing opportunity

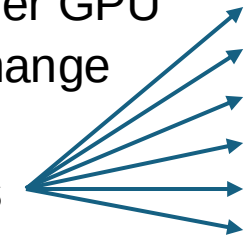
  ⇒ dynamic reconfiguration



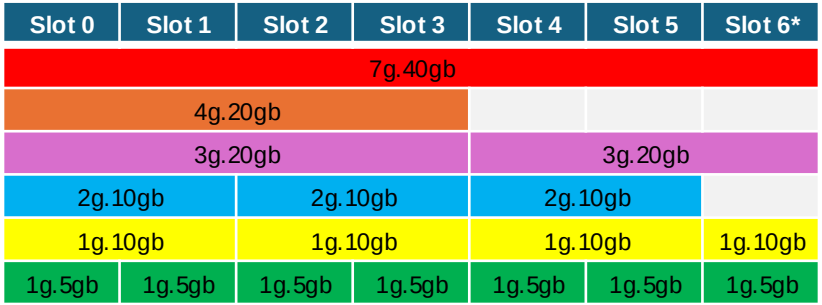


Multi-Instance GPU

- Partition GPU
  - application sees a smaller GPU
  - full isolation, no code change
  - up to 7 slices
  - small number of profiles

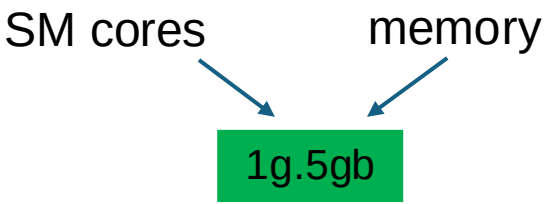


Slot 6 has twice the amount of memory



A100-40GB MIG profiles

- *profiles can be mixed*
- *incremental slice creation/deletion*



Example A100-40GB MIG layouts

MIG via NVIDIA GPU Operator

- Admin label nodes

```
labels:
  nvidia.com/mig.config=all-balanced
```

- MIG manager slices GPUs
- Nodes offer extended resources

```
allocatable:
  nvidia.com/mig-1g.5gb: 2
  nvidia.com/mig-2g.10gb: 1
  nvidia.com/mig-3g.20gb: 1
```

- Pods request these resources

```
resources:
  limits:
    nvidia.com/mig-1g.5gb: 1
```

- Pros
  - stable Kubernetes feature
  - no overlapping slices by design
  - dynamic reconfiguration
    - `nvidia.com/mig.config=all-2g.10gb`
- Cons
  - no partial reconfiguration
    - all workloads are evicted from GPU
  - no incremental configuration
  - slow reconfiguration

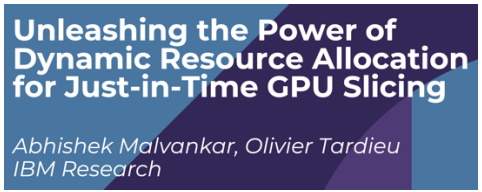
MIG via Dynamic Resource Allocation (Classic)

CLOUD NATIVE & KUBERNETES AI DAY NORTH AMERICA

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
spec:
  restartPolicy: Never
  containers:
  - name: ubuntu
    image: quay.io/quay/ubuntu
    command: ["sh", "-c", "nvidia-smi -L"]
    resources:
    # limits:
    #   nvidia.com/mig-1g.5gb: 1
      claims:
      - name: mig-1g-5gb
  resourceClaims:
  - name: mig-1g-5gb
    source:
      resourceClaimTemplateName: mig-1g.5gb
```

```yaml
apiVersion: resource.k8s.io/v1alpha2
driverName: gpu.resource.nvidia.com
kind: ResourceClass
metadata:
  name: gpu.nvidia.com
---
apiVersion: gpu.resource.nvidia.com/v1alpha1
kind: MigDeviceClaimParameters
metadata:
  name: mig-1g.5gb
spec:
  profile: 1g.5gb
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: mig-1g.5gb
spec:
  resourceClassName: gpu.nvidia.com
  parametersRef:
    apiGroup: gpu.resource.nvidia.com
    kind: MigDeviceClaimParameters
    name: mig-1g.5gb
```

KubeCon CloudNativeCon Europe 2024



https://sched.co/1YeO0

CLOUD NATIVE & KUBERNETES AI DAY NORTH AMERICA

InstaSlice Operator

https://github.com/openshift/instaslice-operator

InstaSlice Operator

- InstaSlice creates MIG slices on demand as required by workloads
  - InstaSlice consumes stable APIs (Kubernetes, NVIDIA)
  - InstaSlice gates pods and works before pods reach the scheduler

- Goals
  - allocate and configure slices
  - account for resource requirements (CPU, memory) when placing GPU slices
  - bind containers to desired slices (on desired nodes and GPUs)
  - release and unconfigure slices when pods terminate or are deleted
  - minimize scheduling latency for inference use cases
  - support pluggable, optimized placement policies
  - integrate with Kubernetes quotas and Kueue quotas

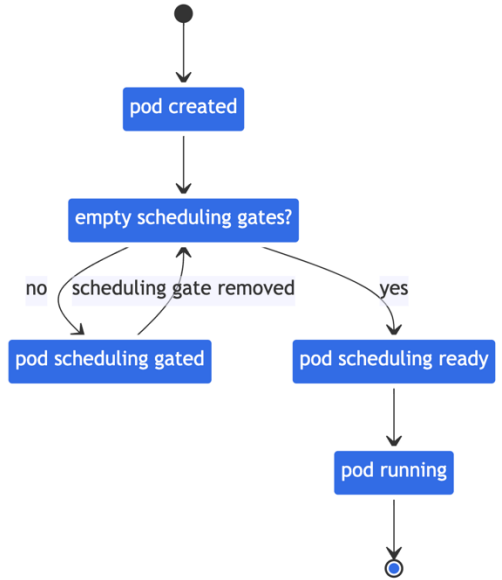
GPU Operator

- Kubernetes provides access to special hardware through device plugin framework
- NVIDIA GPU operator uses operator framework to automate GPU management on Kubernetes
- GPU operator (v24.6.1) is enabled to work with MIG creation and deletion by external controllers
- InstaSlice consumes such a feature of NVIDIA GPU operator

Pod Scheduling Gates

- Pod Scheduling gates is a stable feature in Kubernetes v1.30 that controls when a pod is ready for scheduling
- Use cases
  - Quota Management
  - InstaSlice
    - Wait for MIG slice and associated resources to be provisioned before considering the pod to be scheduled

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  schedulingGates:
  - name: kueue.x-k8s.io/admission
  - name: instaslice.redhat.com/instaslice
  containers:
  - name: busybox
    image: registry.k8s.io/busybox
```



Ref: https://kubernetes.io/docs/concepts/scheduling-eviction/pod-scheduling-readiness/

Placement

- Placement is defined as the act of selecting
  - A node from the cluster and
  - Desired index (start position and size of profile) from multiple GPUs available on the node

- Valid placement should guarantee slice existence, creation and finally consumption by the pod

- Placement in InstaSlice is achieved by
  - Node selectors to send a pod to a node chosen by the controller
  - Mutating the pod to load environment from configmap
    - contents of configmap is MIG partition UUID created dynamically for the pod
    - In later releases we plan to move to CDI for environment var injection

UUID inside uniquely named configmap that selects MIG slice from multiple GPUs available on the node.

```
Name:         accf3112-0789-4662-bc37-8aad54327057
Namespace:    demo
Labels:       <none>
Annotations:  <none>

Data
====
CUDA_VISIBLE_DEVICES:
----
MIG-515999b5-aee0-5631-ad4b-ccc2cf472d95
NVIDIA_VISIBLE_DEVICES:
----
MIG-515999b5-aee0-5631-ad4b-ccc2cf472d95

BinaryData
====

Events:  <none>
```

Architecture

Mutates pod
- Scheduling gate
- Finalizer
- Accelerator resources instaslice.redhat.com

Accounting
- Placement (Node, GPU and GPU index) via
  - node labels
  - configmap

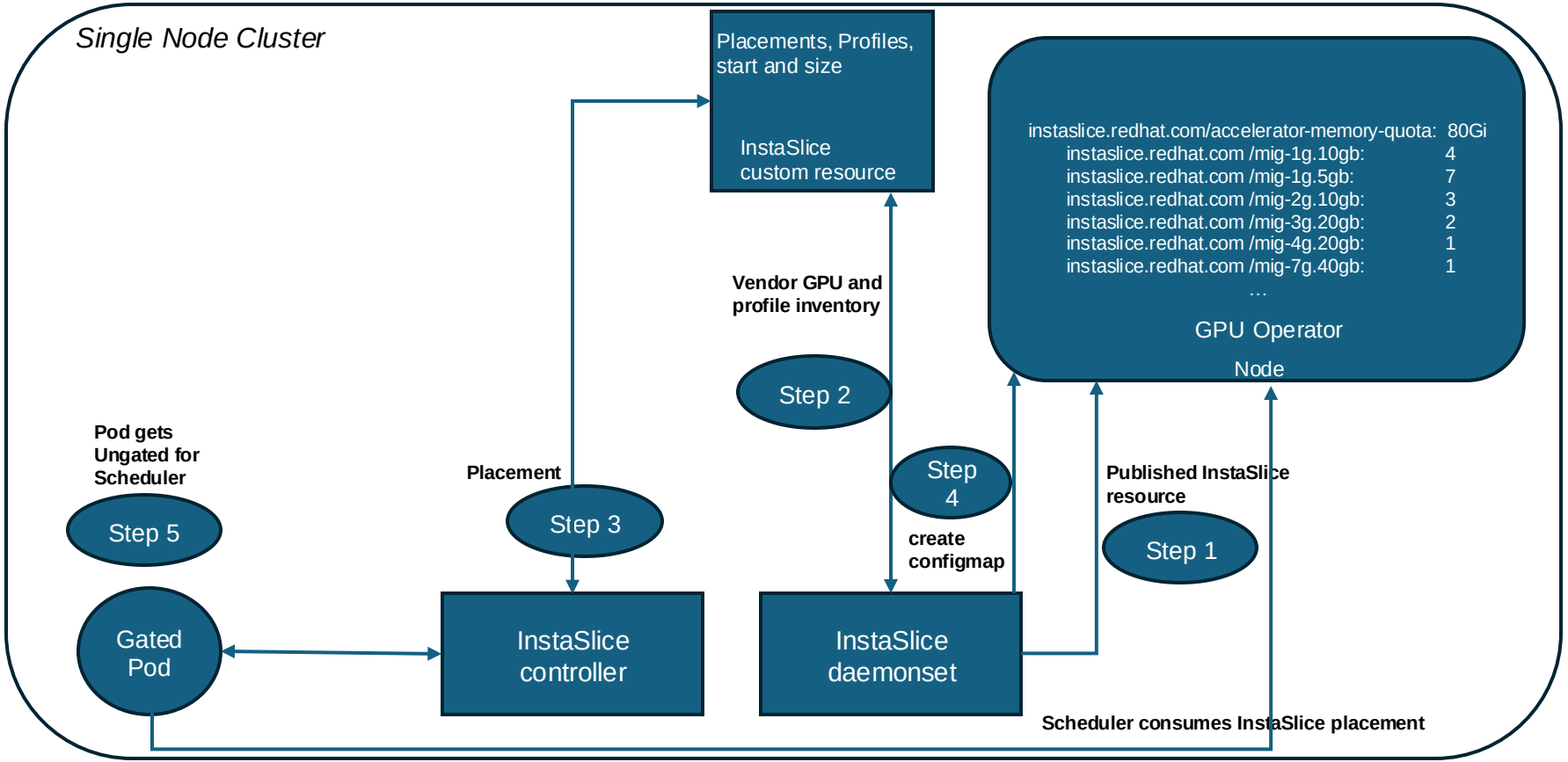
- **Discovers** GPU profiles and populates InstaSlice object
- **Creates** slices via vendor GPU calls with placement
- **Populates** configmap

- Workload gets **assigned** to the controller chose node/GPU and GPU index

States

- InstaSlice custom resource has states that are changed by controller and daemonset to satisfy gated pod intent of creating a slice and accessing it using finalizer.
- Pod enters the system with Kubernetes state as
  - *SchedulingGated*
    - controller picks up such pod and adds it to InstaSlice CR with state as creating
    - daemonset picks up allocations assigned to itself and creates slices on vendor hardware and moves state to created
    - controller ungates the pod that have allocations are created and moves state further to ungated
  - *Pending*
    - Scheduler is consuming InstaSlice placement and binding the pod to the node
  - *Running*
    - InstaSlice allocation stays in state ungated
  - *Terminating*
    - controller sets allocation status to deleting
    - daemonset deletes slices on vendor hardware and moves state to deleted
    - controller removes finalizer and pods finally gets removed by Kubernetes

System Flow


Single Node Cluster
Placements, Profiles, start and size
InstaSlice custom resource
instaslice.redhat.com/accelerator-memory-quota: 80Gi
instaslice.redhat.com /mig-1g.10gb: 4
instaslice.redhat.com /mig-1g.5gb: 7
instaslice.redhat.com /mig-2g.10gb: 3
instaslice.redhat.com /mig-3g.20gb: 2
instaslice.redhat.com /mig-4g.20gb: 1
instaslice.redhat.com /mig-7g.40gb: 1
...
GPU Operator
Node
Vendor GPU and profile inventory
Step 2
Pod gets Ungated for Scheduler
Step 5
Placement
Step 3
Step 4
create configmap
Published InstaSlice resource
Step 1
Gated Pod
InstaSlice controller
InstaSlice daemonset
Scheduler consumes InstaSlice placement

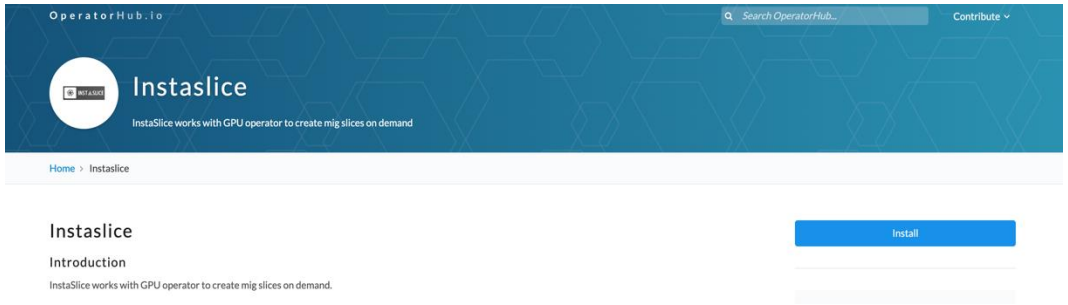
Placement Policy

- To reduce fragmentation and increase pod density on GPU placement becomes important.
- Based on your use cases InstaSlice provides ability to place pods on accelerator tailored to your use case.
- Different placement strategies could be experimented with InstaSlice

```
type AllocationPolicy interface {
  SetAllocationDetails(..)
}
type RightToLeftPolicy struct{}
type LeftToRightPolicy struct{}
type FirstFitPolicy struct{}
```

OperatorHub

- InstaSlice is available on OperatorHub at:
  - https://operatorhub.io/operator/instaslice-operator



CLOUD NATIVE & KUBERNETES AI DAY NORTH AMERICA

InstaSlice Demo

https://youtu.be/W9Kn-cMpokw

Fungible Quotas

- MIG offers choices
  - 7x 1g.5gb slices *or* 1x 7g.40gb slice *or* ...
- Kubernetes quotas are additive
  - 7x 1g.5gb slices *and* 1x 7g.40gb slice *and* ...
- InstaSlice supports fungible quotas
  - webhook maps slice profiles to accelerator-memory-quota quantities
  - daemonset adds accelerator-memory-quota capacity to nodes at startup time

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mig-quota
spec:
  hard:
    nvidia.com/mig-1g.5gb: 7
    nvidia.com/mig-7g.40gb: 1
```

```
nvidia.com/mig-2g.10gb: 1
```

InstaSlice webhook

```
instaslice.redhat.com/accelerator-memory-quota: 10G
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: one-gpu-quota
spec:
  hard:
    instaslice.redhat.com/accelerator-memory-quota: 40G
```

Kueue Integration

- Kueue is a queuing and quota manager for batch workloads
  - workloads can be Pods or Jobs (PyTorchJobs, JobSets, AppWrappers...)
  - https://kueue.sigs.k8s.io
- Kueued workloads are admitted if under quota
  - kueued pods are gated until admitted

  ⇒ InstaSlice ignores gated pods
  - admission decisions for jobs happen *before* pod creation

  ⇒ we add resource transformations to Kueue v0.9.0 to support fungible quotas

```
resources:
  transformations:
  - input: nvidia.com/mig-1g.5gb
    strategy: Replace
    outputs:
      instaslice.redhat.com/accelerator-memory-quota: 5G
```

InstaSlice and DRA

Capabilities	InstaSlice	DRA (v1.32)
Partitionable devices	available today for NVIDIA GPUs	Kubernetes Enhancement Proposal (KEP)
No pod spec changes	available today	under discussion
Fungible quotas	available today	under discussion
Kueue integration	available today	under discussion
Priorities, preemption, autoscaling	planned	under discussion
Pluggable placement policies	planned	under discussion
Shared devices (same slice for 2 pods)	not supported	yes
Multiple devices for one pod	not supported	yes
CPU/memory availability check	approximate	exact
Device selection (H100 vs. A100)	basic (node labels and selectors)	advanced (CEL expressions)

Take Away

- Long-term InstaSlice intends to leverage and complement DRA

- InstaSlice permits incremental GPU slicing *today*
  - with a focus on doing *just that*

- InstaSlice is easy to deploy and use
  - open-source, Apache 2.0 license
    - https://github.com/openshift/instaslice-operator
    - https://operatorhub.io/operator/instaslice-operator
  - same Kubernetes scheduler, same NVIDIA GPU operator
    - simple configuration change to GPU operator
  - same pod specs, same workloads

GitHub

OperatorHub

Team InstaSlice

- Abhishek Malvankar
- Alaa Youssef
- Cameron Meadors
- Harshal Patil
- Kevin Hannon
- Mohammed Abdi
- Mrunal Patel
- Olivier Tardieu
- Ryan Phillips
- Sai Ramesh Vanka
- Vitaliy Emporopulo

Names arranged in alphabetical order

Follow-up Talk

https://sched.co/1i7oh

Thursday November 14 – 3:25pm – 255 B

Please scan the QR Code above to leave feedback on this session