MPS Support in the Kubernetes GPU Device Plugin

Evan Lezar <elezar@nvidia.com>

Last Updated:
May 2, 2024

Table of Contents

Overview

One of the goals of a cluster administrator is to optimize the use of GPU resources. This may include sharing more powerful devices between multiple users to serve use cases which do not require the full memory of computational resources offered by such devices. This can be achieved at a hardware level by partitioning a MIG-capable GPU into MIG devices, or by oversubscription using time-slicing.

The NVIDIA GPU device plugin already supports sharing GPUs via time-slicing and MIG. This document describes an extension to the NVIDIA GPU device plugin which provides the ability to share GPUs using MPS. This allows multiple users to access the same hardware while providing guarantees in terms of available compute and memory resources that are not possible with time-slicing.

The use cases focused on here are limited to those of a cluster administrator partitioning the devices available on a system. Since device plugins do not provide controlled sharing, and an end user has no control over which devices are allocated, use cases where an end-user alone defines the memory or thread limits at an application level are not considered.

In order to allow an administrator to control the sharing of devices using MPS, the sharing config introduced to allow time-slicing is extended with options for MPS as discussed in the following section.

Configuring GPUs with MPS

The following document introduced a new configuration file for use by the k8s-device-plugin and gpu-feature-discovery.

Using a config file to configure the k8s-device-plugin and gpu-feature-discovery

To support MPS with GPUs, we extend this configuration file with the following fields:

```
version: v1
sharing:
  mps:
    renameByDefault: <bool>
```

```
    # failRequestsGreaterThanOne: true
    resources:
    - name: <resource-name>
      replicas: <num-replicas>
    ...
```

That is, for each named resource under sharing.mps.resources, a number of replicas can now be specified for that resource type. These replicas represent the number of MPS-partitions that will be created for each GPU represented by that resource type.

If a GPU is shared with MPS (i.e. a number of replicas greater than 1 is specified), the active thread percentage and default pinned device memory for the device are calculated as follows:

- `activeThreadPercentage = floor(100 / replicas)`
- `pinnedMemoryLimit = floor(totalMemory / replicas)`

As is the case with time-slicing, if `renameByDefault=true`, then each resource will be advertised under the name `<resource-name>.shared` instead of simply `<resource-name>`.

Note that currently `failRequestsGreaterThanOne=true`, is implied and cannot be set to false. This means that the plugin will fail to allocate any shared resources to a container if they request more than one. The container's pod will fail with an `UnexpectedAdmissionError` and need to be manually deleted, updated, and redeployed. This is also the recommended setting to use for `sharing.timeSlicing.failRequestsGreaterThanOne`.

When applying the following MPS sharing configuration to the GPU Device Plugin and GPU Feature Discovery:

```
version: v1
sharing:
  mps:
    resources:
    - name: nvidia.com/gpu
      replicas: 10
```

Each available GPU will be exposed as 10 equal space-partitions each with access to 10% of the total device memory and 10% of the compute resources.

The following labels will be generated by GPU Feature Discovery:

```
nvidia.com/mps.capable = true
nvidia.com/gpu.sharing-strategy = mps
nvidia.com/gpu.product = <product-name>-SHARED
nvidia.com/gpu.replicas = 10
```

As is the case with sharing using time-slicing, the `-SHARED` suffix to the product name can be used in node-selectors to ensure that jobs land on shared GPUs if required.

Furthermore, if the `renameByDefault` option is set to true in the config as below:

```
version: v1
sharing:
  mps:
    renameByDefault: true
    resources:
    - name: nvidia.com/gpu
      replicas: 10
```

The resource name will be changed to `nvidia.com/gpu.shared` and the product name label will not be modified. This means that the following labels will be generated:

```
nvidia.com/mps.capable = true
nvidia.com/gpu.shared.sharing-strategy = mps
nvidia.com/gpu.shared.product = <product-name>
nvidia.com/gpu.shared.replicas = 10
```

Since the resource exposed by the GPU Device Plugin is also renamed to `nvidia.com/gpu.shared`, users can directly request shared resources by updating their pod specs.

Deploying the GPU Device Plugin with MPS support

When using the GPU Operator

At present, there is no integrated support for enabling MPS with the GPU operator. In order to test MPS support with the operator, you will need to deploy it with the device plugin and GFD disabled, as seen below.

```
$  helm install \
      -n gpu-operator \
      --generate-name \
      --create-namespace \
      --set devicePlugin.enabled=false \
      --set gfd.enabled=false \
      nvidia/gpu-operator
```

This will tell the operator not to deploy the device plugin and GFD, nor manage their lifecycle as part of the operator itself. With this in place, you can now follow the instructions below to deploy the helm chart for the device plugin and enable MPS for it separately.

Deploying using Helm

Assuming that the NVIDIA Device Plugin helm repo has been configured, we create a config file that enables MPS sharing for all available devices:

```
cat << EOF > /tmp/dp-mps-10.yaml
version: v1
sharing:
  mps:
    resources:
    - name: nvidia.com/gpu
      replicas: 10
EOF
```

We then deploy the device plugin using this config. Note that since we specify `--set gfd.enabled=true`, GFD is also deployed.

```
helm upgrade -i nvdp nvdp/nvidia-device-plugin \
    --version=0.15.0 \
    --namespace nvidia-device-plugin \
    --create-namespace \
    --set gfd.enabled=true \
    --set config.default=mps10 \
    --set-file config.map.mps10=/tmp/dp-mps-10.yaml
```

When checking the node labels, we note that the labels discussed above are present on the node:

```
$ kubectl get node k8s-device-plugin-cluster-worker  --output=json
| jq '.metadata.labels' | grep -E "mps|SHARED|replicas" | sort
  "nvidia.com/gpu.product": "Tesla-V100-SXM2-16GB-N-SHARED",
  "nvidia.com/gpu.replicas": "10",
  "nvidia.com/gpu.sharing-strategy": "mps",
  "nvidia.com/mps.capable": "true"
```

Furthermore, the presence of the `nvidia.com/mps.capable=true` label triggers the creation of a daemonset to manage the MPS control daemon.

```
$ kubectl get pods -n nvidia-device-plugin -o
custom-columns=NAME:.metadata.name
NAME
nvdp-node-feature-discovery-master-99bd6c9f-fnbpr
nvdp-node-feature-discovery-worker-rb7ps
```

```
nvdp-nvidia-device-plugin-gpu-feature-discovery-5d86g
nvdp-nvidia-device-plugin-mps-control-daemon-2wnjj
nvdp-nvidia-device-plugin-v5f2w
```

Deploying a Workload

First, let's create a namespace for the demos that we're deploying:

```
$ kubectl create ns demo
```

Now, we deploy a container that sleeps to run some sanity checks

```
$ cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Pod
metadata:
  namespace: demo
  name: mps-env-test
spec:
  runtimeClassName: nvidia
  restartPolicy: OnFailure
  containers:
  - name: mps-env-test
    image:
nvcr.io/nvidia/k8s/cuda-sample:nbody-cuda11.7.1-ubuntu18.04
    command: ["sleep", "9999"]
    resources:
      limits:
        nvidia.com/gpu: 1
EOF
```

And confirm that the container is able to communicate with the MPS control daemon:

```
$ kubectl exec -ti -n demo mps-env-test -- bash -c "echo
get_default_active_thread_percentage | nvidia-cuda-mps-control"
10.0
```

And confirm that the container is able to communicate with the MPS control daemon:

```
$ kubectl exec -ti -n demo mps-env-test -- bash -c "echo
get_default_device_pinned_mem_limit 0 | nvidia-cuda-mps-control"
1G
```

(Note that the output for the memory limit is device specific and may output rounded and truncated)

When starting a compute workload as follows:

```
$ cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Pod
metadata:
  namespace: demo
  name: mps-nbody
spec:
  restartPolicy: OnFailure
  containers:
  - name: mps-nbody
    image:
nvcr.io/nvidia/k8s/cuda-sample:nbody-cuda11.7.1-ubuntu18.04
    args: ["--benchmark", "--numbodies=4226048"]
    resources:
      limits:
        nvidia.com/gpu: 1
EOF
```

We see the following output when running nvidia-smi on the host:

```
$ nvidia-smi -i 1
Tue Apr 23 14:00:45 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.105.01   Driver Version: 515.105.01   CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   1  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   40C    P0    95W / 160W |    667MiB / 16384MiB |    100%   E. Process |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    1   N/A  N/A   2525469    M+C   /cuda-samples/sample              637MiB |
|    1   N/A  N/A   2525514      C   nvidia-cuda-mps-server             27MiB |
+-----------------------------------------------------------------------------+
```

Noting the following:

- The GPU has a Compute Mode of Exclusive Process.
- The GPU has an `nvidia-cuda-mps-server` process associated with it.
- The GPU is running the `/cuda-samples/sample` application with type `M+C` indicating that it is communicating through the MPS server process.

Known Issues

Limitations

- The use of the MPS and timeSlicing sharing options are mutually exclusive.
- The maximum number of replicas that can be requested are 16 for pre-Volta devices and 48 for newer devices.
- MPS sharing is not supported for MIG devices.
- Requesting multiple devices when MPS sharing is enabled is not supported.

Revision History

May 2, 2024

- Initial publication

Jun 6, 2024

- Update number of samples in nbody benchmark to 4226048 to remove warning.
  See also: https://github.com/NVIDIA/k8s-dra-driver/pull/129