[YARN-6223] Natively support configuration / discovery / scheduling / isolation for GPUs on Apache Hadoop YARN

Authors: Wangda Tan, Vinod Kumar Vavilapalli
with input from
Varun Vasudev, Sidharta Seethana

Last modified date: Mar 31, 2017

Background

GPUs are increasingly becoming a key tool for many big data applications. Deep-learning / machine learning, data analytics, Genome Sequencing etc all have applications that rely on GPUs for tractable performance.

In many cases, GPUs can get 10x speed ups. And in some reported cases (like this), GPUs can get up to 300x speed ups. Many modern deep-learning applications directly build on top of GPU libraries like cuDNN (CUDA Deep Neural Network library). It's not a stretch to say that many applications like deep-learning cannot live without GPU support.

Before this feature, YARN has a not-so-comprehensive story around GPU support. Today, users use node-labels (YARN-796) to partition clusters to make use of GPUs, which simply put machines equipped GPUs to a different partition and submit jobs that require GPUs to the partition. For a detailed example of this pattern of GPU usage, see Yahoo!'s blog post about Large Scale Distributed deep-learning on Hadoop Clusters.

However, without native and more comprehensive GPU support, there's no isolation of GPU resources. For example, multiple tasks compete for a GPU resource simultaneously which could cause task failures / GPU memory exhaustion, etc.

To this end, we should look for a comprehensive solution to natively support GPU resources on YARN.

Requirements

Challenges

GPU support poses many challenges:

1) There are different levels of resource allocation and isolation, for example access to a whole GPU, to GPU cores, GPU memory and PCI-E/NVLink Bandwidth.
2) For a host with multiple GPUs, topology information is important for performance, which could lead to an order of magnitude difference in latency of interconnection.
3) Running GPU applications inside Docker containers creates even more requirements. For example, it requires versions of drivers in Docker image that exactly match the drivers installed in the host OS.

4) Different vendors (like Nvidia/ATI) expose usage of GPUs in different ways. And different vendors tightly couple the integration with their own drivers and libraries (CUDA/OpenCL).

Requirements

We put the requirements for GPU support to two categories: basic requirements to solve 80% of today's use cases and advanced requirements to solve the rest of the 20% use cases.

Basic requirements

1. (P0)
    a. Use Resource Profiles (YARN-3926) to support specifying GPU as a resource type for scheduling on each node.
    b. Admin is able to config GPU devices in the system which YARN can use.
    c. Granularity of isolation is per GPU device.
    d. Support allocation of N (N < available GPU devices on each node) GPU devices to each task. However, since topology information is not considered by YARN, it's better to allocate one-or-all GPU devices to tasks.
    e. Support GPU isolation in CGroups via devices subsystem, which can be used by both of *LinuxContainerExecutor* and Docker.
    f. Validate support for Nvidia devices, the most common setup for now. Though this isn't a design requirement, one can consider this is a requirement that impacts concrete configurations and testing coverage.
2. (P1)
    a. Support automatic discovery of available GPU devices in the system.
    b. Better GPU on Docker support: With Docker, one can automatically mount GPU related binaries to Docker volume and load kernel module, etc. We may also need additional plugins to discover GPU devices and load volumes, etc.
    c. Expose GPU usage information to monitoring framework (like using JMX).

Advanced requirements

- (P3) Support different versioning of libraries/applications (Like cuDNN >= 5.0, CUDA >= 8.0).
- (P4) Support fine-grained GPU resource isolation and sharing like SM (Streaming-Multiprocessor).
- (P4) Support more GPU resource types like GPU memory / bandwidth.
- (P4) Validate support for more GPU vendors other than Nvidia.
- (P4) Support multiple GPU topology and affinities to devices. (Similar to NUMA support in YARN)

Relation to other features

1) YARN-3926, Extend the YARN resource model for easier resource-type management and profiles:

YARN-3926 is blocker for this feature, we need to recognize GPU as a resource type when doing scheduling.

2) YARN-796, Node partition support in YARN

YARN-796 will be helpful to this feature.

For a cluster which runs GPU workload mixed with other workloads, admins can use ACLs/per-partition-queue-capacities to make sure only some specific queues/applications can access GPU resources.

This can also avoid tasks request GPUs get starved because general resources like memory / CPU on a machine equipped GPUs grabbed by tasks which don't need GPU resources.

3) YARN-3409, Node constraint.

Admins can use this feature to tag node with different attributes. For this feature, admins can tag versions of CUDA(Like CUDA=8.5)/driver(like NVIDIA_DRIVER=378.92) and GPU architecture/series (K80).

Proposal

As the first step, we plan to do the following:

GPU isolation story

Alternatives for per-GPU device resource isolation on Linux

GPU vendors expose GPU devices to a specific path like /dev.

- */dev/gpu-device[minor-number]* are GPU devices.
- Some GPU-related devices are related to kernel. They must be accessible by all processes which require GPU resources.

There are different approaches to isolate per-GPU device resource on Linux platform.

1) Using *devices* module in CGroups

CGroups support devices module which can be used to limit devices' visibility to launched processes by specifying major/minor numbers of devices.

(Please note that, this document says that devices support is still 'tech preview' in RHEL 6 - we'll have to investigate further to see which distributions and which versions of this distributions this can be supported on, in a production-ready manner.)

2) Using *docker --device*

Docker can specify --device option to specify devices which can be accessed by launched containers. For example,

```
docker run --device=/dev/gpu0 ...
```

This only allows the launched container to access GPU-0.

Concrete proposal for isolation

For all the options listed above, we prefer to do GPU isolation via CGroups first. The reason behind the decision is most of today's workloads on YARN are in non-Docker environment. And using CGroups to do isolation is a balanced solution which can be used by both of Docker and non-Docker (LinuxContainerExecutor) setup.

Since motivation behind Docker are hardware-agnostic and platform-agnostic, once we have done isolation via CGroups, we can improve YARN to better support Docker containers which need GPUs.

YARN-3443 added *ResourceHandler* abstraction to YARN which makes easier adding new resource isolation module for new resource types. We can create new class (Like *GpuResourceHandler*) which implements the ResourceHandler.

To enable GPU isolation, admin need to set following configuration in yarn-site.xml

```
yarn.nodemanager.resource.gpu.enabled=true
```

GPU resource discovery story

For properly doing scheduling and isolation, we need to know:

1. How many GPU devices are available in the system?
2. What are the minor numbers of these GPU devices?

Building tools to get GPU devices and minor numbers (Like using NVML) is simple but it will unavoidably add new build dependencies to the system. To start with, we propose to let admin configure these parameters in yarn-site.xml and later on provide tools to do automatically discovery of GPU devices.

In addition to that, admin can control only letting YARN to manage subset of GPU devices available on each host.

Admin can enable following configurations in yarn-site.xml

```
# Allow YARN to manage GPU devices with minor number 0 and 2
yarn.nodemanager.resource.gpu.allowed-gpu-devices = 0,2
```

For YARN NodeManager, it will use the above configured minor number to blacklist GPU devices used by launched process. Like:

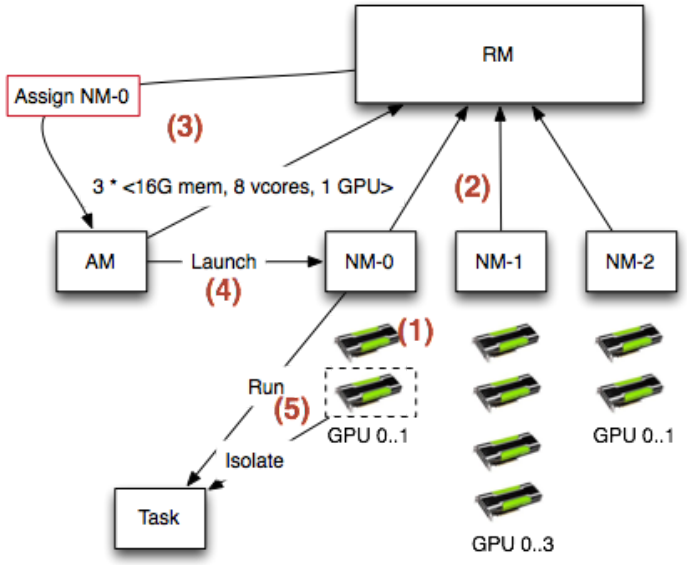
```
echo “c {major-number}:{minor-number} rwm” >
${cgroups-mount-path}/devices/devices.deny
```

Notes:

1) ${cgroups-mount-path} is the same path that YARN uses to use cgroups to limit container on other resources: Admin needs to configure *yarn.nodemanager.linux-container-executor.cgroups.mount-path* in yarn-site.xml for it.
2) The reason for using the “blacklisting” approach instead of “whitelisting” is the following: Usually, there are some kernel controller devices which have the same major number but different minor numbers. By default, processes launched in cgroups can access all devices, subject to the device / file-system permissions. If we want to support whitelisting, we need to populate in the cgroup all the devices that the process can access, starting with the minor numbers for devices that should be universally accessible. This will be harder than simply blacklisting devices which YARN doesn’t want processes to access.
3) If a container doesn’t need GPU launched on a GPU host, YARN should blacklist all GPU devices for the container.
4) YARN should properly recover information of assigned GPUs to containers upon NodeManager restart.

Implementation details

Workflow



This feature works end-to-end in following way:

1) Each NodeManager configures GPU devices number which YARN can use. (Either via admin configuration or automatically GPU device discovery).
2) NodeManagers report GPU resources along with other resources (Like CPU/memory) to RM.
3) When application makes resource-requests (for e.g. 3 of <16G mem, 8 vcores, 1 GPU>) to RM, scheduler allocates requested resource on NodeManagers and responds to AM.
4) AM talks to NodeManager to launch the task on allocated container.
5) NodeManager launches the container and enforces GPU resources along with other resources.

Specific vendor example

Let's take Nvidia as an example to see about implementation.

GPU resource configuration and discovery

The answers to #1 is relative simple, using Nvidia built-in tool *nvidia-smi* can directly get it, output of *nvidia-smi* looks like:

```
GPU  Name        Persistence-M
```

```
0  GeForce GTX 760     Off

Fan  Temp  Perf  Pwr:Usage/Cap
0%   34C  N/A     N/A /  N/A
```

However, according to this, GPU number showing in nvidia-smi output could be different from minor number.

To be able to get minor numbers of GPUs, *nvmlDeviceGetCount* and *nvmlDeviceGetMinorNumber* of NVML (Nvidia Management Library) can be used to achieve this.

GPU device isolation via CGroups

For Nvidia GPU devices, YARN can write major numbers of Nvidia (195) with minor number (like 0/1/2 …) to devices.deny. For example

```
echo “c 195:0 rwm” > ${cgroups-path}/devices/devices.deny
```

This prevent the process to access *the GPU with minor number 0*.

GPU device isolation via docker

Support GPU isolation via docker can be done in two ways (a) directly passing devices to docker and implementing any additional functionality needed (b) use nvidia-docker (license)

For (a), we can use Nvidia exported GPU devices at */dev/nvidia[minor-number]* in additional to the kernel level devices */dev/nvidia-ctl*, */dev/nvidia-uvm*.

Nvidia-docker integrated many good functionalities like searching and mounting Nvidia libraries, etc. It will be too complex to re-implement *nvidia-docker* on YARN. Instead, we can directly leverage nvidia-docker to do next level of GPU support of Docker containers in short term..

Nvidia-docker is a wrapper of docker, created by Nvidia to leverage Nvidia GPU devices when running docker containers. For example,

```
nvidia-docker run -ti nvidia/cuda nvidia-smi
```

Like mentioned above, in addition to GPU isolation, *nvidia-docker* can properly mount required libraries to Docker container which makes image agnostics to Nvidia Driver. More details please refer to Why NVIDIA Docker. And GPU Isolation.

Open questions

1) How to do GPU resource isolation on Windows?
2) How to support other GPU vendors like ATI?
3) How to enforce sharing of multiple tenants running on a single GPU device.