Univa Corporation

Using NVIDIA® DGX™ Systems with Univa Grid Engine

Table of Contents

1. Introduction

NVIDIA® DGX™ Systems are a family of products from NVIDIA purpose-built for Deep Learning applications. The NVIDIA DGX family is comprised of the NVIDIA DGX Station™, and NVIDIA DGX-1™ and DGX-2™ rackmount servers. The NVIDIA DGX-2 servers provide up to 16 NVIDIA Volta™ V100 GPUs with an NVIDIA NVSwitch™ powered NVLink™ fabric for superior performance.

In AI data centers, managing GPU-aware distributed machine learning software frameworks is a central challenge. Data scientists run diverse workloads that range from preparing datasets to model training to ongoing validation. Workloads need to run quickly, use resources efficiently, and be deployed intelligently, considering factors such as CPU and GPU architectures, memory, cache, server bus topologies, and NVIDIA interconnect and network switch topologies.

Univa Grid Engine is an enterprise-class workload scheduling and management solution used across many industries for applications that include machine learning and deep learning. Univa Grid Engine provides rich support for scheduling GPU-aware applications and containers. It also features a direct integration with NVIDIA Data Center GPU Manager (DCGM) making it an ideal workload manager for NVIDIA DGX environments.

This document provides a quick start guide for users configuring Univa Grid Engine for use with NVIDIA DGX systems.

2. Obtaining Univa Grid Engine

You can obtain a copy of Univa Grid Engine by contacting Univa at http://univa.com or requesting a free software trial from the Univa website at http://www.univa.com/resources/univa-grid-engine-trial.php.

Please note that there are open-source versions of Grid Engine available, but open-source Grid Engine does not include the GPU-aware scheduling features required for NVIDIA DGX systems.

3. Before you start

Univa Grid Engine is typically installed on a cluster comprised of multiple NVIDIA DGX systems connected by a TCP/IP network. Univa Grid Engine should be installed by a Linux system administrator familiar with the NVIDIA DGX server environment.

You will need to perform some prerequisite steps before installing Univa Grid Engine. These include selecting a master host, ensuring that the TCP/IP network is properly configured, and setting up a shared NFS file system accessible to NVIDIA DGX hosts. Consult the Univa Grid Engine Installation Guide for details.

Univa Grid Engine will work the Ubuntu® LTS operating system pre-installed on NVIDIA DGX systems. Univa Grid Engine is also supported on most other Linux operating systems including Red Hat® Enterprise Linux®, SUSE® Linux® Enterprise Server, CentOS®, and Oracle® Linux®. Details of the supported operating systems can be found in the Univa Grid Engine Release Notes.

These instructions assume that you are running Univa Grid Engine 8.6.5 or later. The steps described in sections 4 to 6 of this guide are essential for configuring GPU and DCGM support in Univa Grid Engine on DGX systems. Setting up Docker as described in section 7 and CPU-GPU affinity as described in section 9 are optional.

4. Enabling cgroups

Cgroups (control groups) is a feature available in modern Linux distributions that limits the resources that a process or set of processes can have access to.

Univa Grid Engine supports multiple features related to cgroups. The `qconf` command is used by Univa Grid Engine administrators to display or modify the configuration of the cluster. Configuration changes can either be made globally such that they apply to all cluster hosts, or individually. Individual host settings will override the global settings.

On NVIDIA DGX systems cgroups needs to be enabled by setting the `cgroup_path` and the `devices` parameters so that Univa Grid Engine will manage access to all GPUs on the host. This will block all devices from /dev/nvidia0 to /dev/nvidia254 and make them unavailable to users and processes running outside of Univa Grid Engine.

To run `qconf` you will need to be logged in as the Univa Grid Engine administrator account specified when you installed Univa Grid Engine. You can run `qconf` from any cluster host.

When you run `qconf`, you will be placed in the `vi` editor (or the editor pointed to by $EDITOR). You will need to add or modify the `cgroup_params` line as shown[1] below using `vi`. After making changes, you will need to save them in `vi` to return to the shell.

```
$ qconf -mconf <hostname>

cgroup_params  cgroup_path=/sys/fs/cgroup \
               devices=/dev/nvidia[0-254]
```

A DGX host won't have 255 physical GPUs. This setting ensures that Univa Grid Engine will manage all the GPUs.

[1] These instructions apply to Univa Grid Engine 8.6.5 or later. If you are running an earlier version of please check the documentation to learn how to configure the devices setting in cgroup_params.

5. Enabling DCGM support

Univa Grid Engine 8.6.0 and later is integrated with NVIDIA's Data Center GPU Manager (DCGM) providing detailed information about GPU resources. With this integration, Univa Grid Engine has full visibility to GPU details on each host including the GPU type and version, available memory, operating temperature, and socket, core and thread affinity. This information helps Univa Grid Engine schedule GPU-aware applications more efficiently to optimize both performance and resource usage.

Univa Grid Engine runs a daemon on each cluster host called `execd`. DCGM support is enabled by setting the execd parameter `UGE_DCGM_PORT` to the port that DCGM uses to communicate on each host, 5555 by default.

You can run the `qconf` command below as the Univa Grid Engine administrator to set the `UGE_DCGM_PORT` for each cluster host where DCGM is installed.

```
$ qconf -mconf <hostname>

execd_params    UGE_DCGM_PORT=5555
```

If all hosts in your cluster have GPUs and DCGM installed, you can run the command below to apply this setting globally to all cluster hosts.

```
$ qconf -mconf global

execd_params    UGE_DCGM_PORT=5555
```

6. Setting up GPU resources

Univa Grid Engine uses a Resource Map construct called an RSMAP complex to manage access to "consumable" resources on a host. Consumable resources are an efficient way of managing limited resources that are consumed, such as memory, free space on a file system, floating software licenses or GPUs.

Before RSMAPs can be used, a consumable resource of type RSMAP must be added in Univa Grid Engine as shown. As before, the `qconf` command should be run as the cluster administrator and `vi` will be used to enter or edit parameter settings. Details about what these settings mean are provided in the `sge_complex` main page.

```
$ qconf -ace gpu

name        gpu
shortcut    gpu
```

```
type        RSMAP
relop       <=
requestable YES
consumable  YES
default     0
urgency     0
aapre       NO
affinity    0.000000
```

Each id of the RSMAP complex can be configured to represent a device on the host indicated. The “device” parameter is needed to facilitate blocking using cgroups. The `cuda_id` provides the mapping between the device as it appears in DCGM and physical device visible to Univa Grid Engine. The same `cuda_id` is used by `nvidia_smi` also.

The `qconf -me` command is used to modify the configuration of each execution host. For an NVIDIA DGX-2 system with 16 GPUs the configuration looks like this:

```
$ qconf -me dgx2-1

hostname          dgx2-1
...
complex_values    gpu=16(gpu0[device=/dev/nvidia0,cuda_id=0] \
                  gpu1[device=/dev/nvidia1,cuda_id=1] \
                  gpu2[device=/dev/nvidia2,cuda_id=2] \
                  gpu3[device=/dev/nvidia3,cuda_id=3] \
                  gpu4[device=/dev/nvidia4,cuda_id=4] \
                  gpu5[device=/dev/nvidia5,cuda_id=5] \
                  …
                  gpu15[device=/dev/nvidia15,cuda_id=15])
```

Each GPU can be optionally represented by more than one complex/id. This is discussed in section 9 dealing with oversubscription of GPUs.

7. Installing and configuring Docker

To submit Docker jobs requiring access to GPUs, nvidia-docker must be installed on each cluster host. Docker and nvidia-docker are installed on NVIDIA DGX-1 and DGX-2 systems by default, but if you use a different operating system, you will need to install these components yourself.

You will need a Docker image with Nvidia GPU support containing your application. You can use the NVIDIA supplied cuda:9.0-base or cuda:10.0-base images or other images derived from these base images. There are a variety of nvidia-docker images available at NVIDIA’s GPU Cloud (https://ngc.nvidia.com).

Univa Grid Engine has built-in support for Docker and nvidia-docker2 allowing you to manage containerized GPU workloads just as you would manage any Grid Engine job.

When submitting a nvidia-docker job to Univa Grid Engine, you need to specify the runtime on the Univa Grid Engine `qsub` command line by adding `-xd "--runtime=nvidia"`.

Running a containerized Tensorflow training model under control of Univa Grid Engine to select a specific GPU will look something like this:

```
qsub -l
docker,docker_images="*tensorflow:18.03-
py2*",gpu=1[affinity=true],name="Tesla V100-PCIE-16GB" -xd "--
runtime=nvidia" -b y -S /bin/sh <command-inside-container>
```

To avoid the need to add the `-xd` switch with every job submission you can change the default Docker runtime on each DGX host by changing the contents of `/etc/docker/daemon.json`:

```
{ "runtimes": { "nvidia": { "path": "/usr/bin/nvidia-container-runtime",
"runtimeArgs": [] } }, "default-runtime": "nvidia" }
```

8. Submitting jobs

In this section, we provide some simple examples of running different kinds of jobs under Univa Grid Engine on an NVIDIA DGX cluster. The `qsub` command in Univa Grid Engine is used to submit jobs to the cluster. Details are available in the Univa Grid Engine User's Guide.

8.1 Regular jobs

Non-GPU jobs run on the NVIDIA DGX cluster just as they would on any other Grid Engine cluster. Jobs that attempt to access a GPU at runtime without specifically requesting a GPU will be blocked by cgroups.

Submit an array of five non-GPU tasks numbered 2,4,6,8,10 to a priority queue

```
$ qsub -t 2-10:2 -l q=priority array.sh
```

8.2 GPU-aware jobs

Jobs can request access to a GPU using the `-l` switch on the `qsub` command line to make a resource request. Resource requests can include the resource name, the amount of resources, the name or ID of the resource, and directives related to affinity. The general syntax is: `-l <cplx_name>="<amount>(<id>)[<affinity>]"`.

Request access to one GPU:

```
$ qsub -l gpu=1 gpu_job.sh
```

Request access to a specific GPU (a GPU with the id “V100” must be configured):

```
$ qsub -l gpu=”1(V100)” gpu_job.sh
```

When a user submits a job requesting a GPU as above, Univa Grid Engine responds with a `job_id`. The user can then use the `qstat` command as shown below to get status information about the job including information about the assigned GPU.

```
$ qstat -j <job_id>
exec_host_list         1:    dgx03:1
granted_req.           1:    gpu=1
granted devices        1:    dgx03: /dev/nvidia0
resource map           1:    gpu=dgx03=(V100)
```

8.3 Docker GPU jobs

Docker jobs requiring GPU resources must request a docker host with the required docker image and specify the number of GPUs required for the containerized workload. In the example below, we require two GPUs, a host where the Docker resource is set to true (meaning Docker is installed), and we would prefer a host that already has the required cuda:9 docker image loaded to avoid needing to pull the image again from a repository. Univa Grid Engine has visibility to a list of all docker images available on each host, so a wildcard is used to match a specific image against the list.

The `--env NVIDIA_VISIBLE_DEVICES` argument is specified on the Univa Grid Engine `qsub` command line to pass the environment variable `NVIDIA_VISIBLE_DEVICES` into the container.

The placeholder values `${gpu(0)} and ${gpu(1)}` are replaced by the actual GPU devices scheduled by Univa Grid Engine. The default for `NVIDIA_VISIBLE_DEVICES` is “`all`” so passing only the GPUs selected by Grid Engine ensures that there are no conflicts and workloads cannot interfere with one another.

```
$ qsub -l gpu=2,docker=1,docker_images=”*cuda:9*” \
 -xd "--runtime=nvidia" \
 -xd "--env NVIDIA_VISIBLE_DEVICES=${gpu(0)},${gpu(1)}" \
  gpu_job.sh
```

When the default docker runtime is set to nvidia as explained in section 7, the `-xd "--runtime=nvidia"` directive can be omitted from the command line.

9. Requesting CPU-GPU affinity

With GPU-aware applications, part of the workload runs on one or more CPU cores, and part of it runs on a GPU. It is important that processor cores be close to a selected GPU, sharing a common switch or PCIe bus to maximize performance.

When CPU cores are close to a GPU, they are described as having affinity. Affinity is an important consideration when Univa Grid Engine makes scheduling decisions. Requesting CPU-GPU affinity is only possible when support for DCGM is enabled as described in section 5.

Part of the information that DCGM provides to Univa Grid Engine for each GPU is information about processor affinity. The affinity value in the example below shows the CPU sockets, cores, and threads on the host that have affinity to each GPU by displaying them in upper case. This example shows a dual-CPU host where each processor (socket) has 8 cores each with 2 threads. If a GPU workload is placed on the first GPU (cuda0), the best performance will be obtained by scheduling the CPU component of the job on the first 4 cores on either socket.

```
host.cuda.0.affinity=SCTTCTTCTTCTTcttcttcttcttSCTTCTTCTTCTTcttcttcttc
tt,
host.cuda.0.gpu_temp=36,
host.cuda.0.mem_free=16280.000000M,
host.cuda.0.mem_total=16280.000000M,
host.cuda.0.mem_used=0.000000M,
host.cuda.0.name=Tesla V100-PCIE-16GB,
host.cuda.0.power_usage=28.527000,
host.cuda.0.verstr=390.46,
host.cuda.1.affinity=ScttcttcttcttCTTCTTCTTCTTScttcttcttcttCTTCTTCTTC
TT,
host.cuda.1.gpu_temp=40
..
```

This is complicated and behind the scenes, but DCGM and Univa Grid Engine hide this complexity from the user. Jobs requiring GPUs can request that they are scheduled on a CPU-GPU combination with good affinity by using the optional `affinity` parameter as shown:

```
$ qsub -l gpu="1[affinity=1]" gpu_job.sh
```

These jobs will automatically be bound to CPU cores that have a good affinity to the assigned GPUs. If no CPU-GPU combination is available/free, the job will not be scheduled.

In Univa Grid Engine versions before 8.6.5, the affinity parameter can be 0/false (the default) or 1/true. In Univa Grid Engine 8.6.5 and later, affinity can also be set to "`2`" meaning that affinity is "nice to have". If a job is submitted with `affinity=2` Univa Grid Engine will attempt to schedule CPU cores and GPU devices with good affinity, but will schedule the job anyway if the affinity requirement cannot be satisfied.

10. Learn more

For additional information about using Univa Grid Engine with NVIDIA DGX systems, please consult the following Univa Grid Engine documentation:

- Univa Grid Engine Installation Guide
- Univa Grid Engine Administrator's Guide
- Univa Grid Engine User's Guide

You can contact Univa by visiting univa.com or obtain technical support by sending an email to support@univa.com.